Rick Reed
Attila Bilgic
Reinhard Gotzhein (Eds.)

SDL 2009: Design for Motes and Mobiles

14th International SDL Forum
Bochum, Germany, September 2009
Proceedings

LNCS 5719

Springer

Rick Reed   Attila Bilgic
Reinhard Gotzhein (Eds.)

SDL 2009: Design for Motes and Mobiles

14th International SDL Forum
Bochum, Germany, September 22-24, 2009
Proceedings

Springer

Volume Editors

Rick Reed
Telecommunications Software Engineering
The Laurels, Victoria Road
Windermere, Cumbria LA23 2DL, UK
E-mail: rickreed@tseng.co.uk

Attila Bilgic
Ruhr-Universität Bochum
Chair of Integrated Systems
44801 Bochum, Germany
E-mail: attila.bilgic@is.rub.de

Reinhard Gotzhein
University of Kaiserslautern
Networked Systems Group
67653 Kaiserslautern, Germany
E-mail: gotzhein@informatik.uni-kl.de

Library of Congress Control Number: 2009934524

CR Subject Classification (1998): C.2, D.2, D.3, F.3, C.3, H.4

LNCS Sublibrary: SL 5 – Computer Communication Networks
and Telecommunications

ISSN 0302-9743
ISBN-10 3-642-04553-7 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-04553-0 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12763993 06/3180 5 4 3 2 1 0

Preface

This volume contains the papers presented at the 14th SDL Forum, Bochum, Germany entitled *Design for Motes and Mobiles.* The SDL Forum has been held every two years for the last three decades and is one of the most important open events in the calendar for anyone from academia or industry involved in System Design Languages and modelling technologies. It is a primary conference event for discussion of the evolution and use of these languages. The most recent innovations, trends, experiences, and concerns in the field are discussed and presented. The SDL Forum series addresses issues related to the modelling and analysis of reactive systems, distributed systems, and real-time and complex systems such as telecommunications, automotive, and aerospace applications. The intended audience of the series includes users of modelling techniques in industrial, research, and standardization contexts, as well as tool vendors and language researchers.

Of course, during the last three decades languages, associated methods, and tools have evolved and new ones have been developed. The application domain has changed almost beyond recognition. Three decades ago the mobile technology of today was science fiction, whereas now we find software systems embedded in inexpensive childrens' toys. More recently multi-core processors have become common technology for consumer computers, and are beginning to be applied in small devices. Even in small co-operating, independently powered remote devices (such as motes and mobile phones), there is enough memory and processing power to support quite sophisticated operating systems and applications. No longer do these need to be hand-coded in a machine-level language, and it is cost effective to apply the languages, tools, and methods that previously applied to systems for telephony routing or automated manufacture.

Many programming language support systems assume a single processor, or that distribution of processes over processors is handled by the operating system. System Design Languages such as the Unified Modeling Language or ITU-T Specification and Description Language allow engineers to defer the distribution until later in product engineering, and also to defer whether the distributed components of systems are loosely (or tightly) coupled. However, these issues still need to be tackled, and therefore a focus of SDL 2009 was on multi-processor and multi-core issues. The concern is what impact does development for this kind of system have on the model-driven approach, engineering languages, and operating system support. Papers in this volume address such issues or applications that use motes.

Based on experience with the previous SDL Forum, it was decided to not only call for papers based on well-advanced or established work, but also to invite short papers describing work-in-progress. One submitted paper that appears in this volume is very clearly in the category: "Towards Model-Based Development of Managed Networked Embedded Systems." This work is at an early stage, but the topic is certainly an important one, as we can anticipate further development of MDD and increasing numbers of networked embedded systems with individual components that are even more powerful. The method for developing the management system is the focus of this work. It will be interesting to see the results.

Another paper applies aspect orientation to the User Requirements Notation (URN). This language has only recently reached the status of a standard, and illustrates that it is not just the applications that are changing in nature over the years: URN has extended the range of formal languages to the requirements area, and the proposal to add aspect orientation shows that it is a *living language* that is evolving to user needs. In a few years' time URN, supporting tools, and use of the language will have evolved. In that case URN may be considered the natural way to design products with state-based models being thought of as intermediate languages, in the same way that currently the ITU-T Specification and Description Language is considered as the design with transcompilation into C. All that we can really predict is that after another decade, at the 19^{th} SDL Forum, it is likely that system design will be at a higher level, with more advanced languages, methods, and tools.

Thanks

As always, the event and this volume would not exist without the contributions of authors, who are thanked for their work.

The Programme Committee and Anders Olsen (Cinderella, Denmark) were reviewers of the papers, and are thanked for their work selecting the papers and the programme.

The organization of SDL 2009 was assisted by sponsorship and support from:

– IBM Rational
– Forschungsschwerpunkt "Ambient Systems"
– International Telecommunication Union

July 2009

Rick Reed
Atilla Bilgic
Irv Badr
Reinhard Gotzhein

SDL Forum Society

The SDL Forum Society is a not-for-profit organization that in addition to running the SDL Forum series of events:

- Has usually run[1] the SAM (System Analysis and Modeling) workshop every two years between SDL Forum years.
- Is a body recognized by ITU-T as co-developing the Z.100 to Z.109 and Z.120 to Z.129 and other language standards.
- Promotes the ITU-T System Design Languages.

For more information on the SDL Forum Society, see `www.sdl-forum.org`.

[1] In 2008 there was no SAM workshop, but instead a one-day workshop on System Design Languages was held in collaboration with ITU-T at ITU-T in Geneva.

Organization

Each SDL Forum is organized by the SDL Forum Society with the help of local organizers. The Organizing Committee consists of the Board of the SDL Forum Society plus the local organizers and others as needed depending on the actual event. For SDL 2009 the local organizers from Ruhr-Universität Bochum and hosting sponsor IBM Rational need to be thanked for their effort to ensure that everything was in place for the presentation of the papers in this volume.

Organizing Committee

Chair

Atilla Bilgic	Ruhr-Universität Bochum

SDL Forum Society

Chairman: Rick Reed (TSE)
Treasurer: Martin von Löwis (Hasso-Plattner-Institut)
Secretary: Reinhard Gotzhein (Universität Kaiserslautern)

Hosting Sponsor

Irv Badr	IBM Rational

Programme Committee

Conference Co-chairs

Atilla Bilgic	Ruhr-Universität Bochum, Germany
Reinhard Gotzhein	Universität Kaiserslautern, Germany

Members

Daniel Amyot	Université d'Ottawa, Canada
Irv Badr	IBM Rational, USA
Atilla Bilgic	Ruhr-Universität Bochum, Germany
Rolv Bræk	NTNU, Norway
Laurent Doldi	Isoscope, France
Anders Ek	IBM Rational, Sweden
Stein-Erik Ellevseth	Devotek, Norway
Joachim Fischer	Humboldt-Universität zu Berlin, Germany
Jacqueline Floch	SINTEF, Norway
Emmanuel Gaudin	PragmaDev, France

Table of Contents

Model-Driven Construction of Embedded Applications Based on Reusable Building Blocks – An Example

Frank Alexander Kraemer, Vidar Slåtten, and Peter Herrmann

Norwegian University of Science and Technology (NTNU),
Department of Telematics, N-7491 Trondheim, Norway
`{kraemer,herrmann,vidarsl}@item.ntnu.no`

Abstract. For the rapid engineering of reactive systems we developed the SPACE method, in which specifications can be composed of reusable building blocks from domain-specific libraries. Due to the mathematical rigor and completeness with which the building blocks are designed, we can provide tool support facilitating a high degree of automation in the development process. In this paper, we focus on the design of embedded Java applications executed on Sun SPOTs by providing dedicated blocks to access platform-specific functionality. These building blocks can be used in combination with other blocks realizing protocols such as leader election to build more comprehensive applications. We present an example specification and discuss its automatic verification, transformation and implementation.

1 Introduction

Maybe it is just that engineers still love the LEGO bricks of their childhood, but creating software systems by connecting reusable building blocks seems to be an attractive development paradigm that can facilitate reuse and enable an incremental development style in which problems can be solved block by block. Yet the everyday practice by developers often does not work as smoothly as simply plugging together bricks: Major challenges lie in the nature of reusable modules in the first place, especially in how to encapsulate and how to compose them. Our engineering method SPACE [1,2] aims to address these issues. As reusable units we use special building blocks that express their behavior in terms of UML activities. These can be composed by pins, and a system can be constructed as a hierarchy of building blocks. While building blocks can describe local behavior executed by a single component, they can in general also cover *collaborative* behavior among *several* components. This facilitates the reuse of solutions to problems that require the coordination of several components, and is especially useful to describe services.

While our method is general and useful in a variety of domains, we demonstrate in this article its application in the area of embedded systems. For that, we present the results of a case study on a sensor network carried out as part of the applied research project ISIS[1] (Infrastructure for Integrated Services [3]), in which we develop methods, platforms and tools for the model-driven development of reactive systems for applications in home network systems. The case study is implemented on small processing devices from Sun Microsystems, called *Sun SPOTs* [4] that run Java.

In the following, we cover all steps needed to realize deployable code from high-level specifications. We will focus especially on the definition of building blocks for the domain of Sun SPOTs and on a protocol for fault-tolerant leader election. We start with an introduction of Sun SPOTs including the runtime support system, followed by a brief overview of our method. In Sect. 2, we present the example system and its high-level specification based on UML activities. The next two sections document our library for Sun SPOTs and the leader election algorithm. In Sect. 5 and 6, explanations of the automated analysis and implementation follow, in which state machines similar to SDL processes are synthesized, from which code is generated.

1.1 Embedded Java on Sun SPOTs

A sketch of a Sun SPOT is shown on the left side of Fig. 1. Each SPOT is equipped with two buttons and sensors for temperature, light and acceleration. SPOTs can also carry extension cards to interact with various other devices. A Sun SPOT is controlled by a 32-bit ARM 9 processor that can run the Java virtual machine *Squawk* [5] executing Java 1.3 code following the CLDC 1.1 specification. SPOTs can communicate among each other using IEEE 802.15.4 radio communication, and build a mobile ad hoc network.

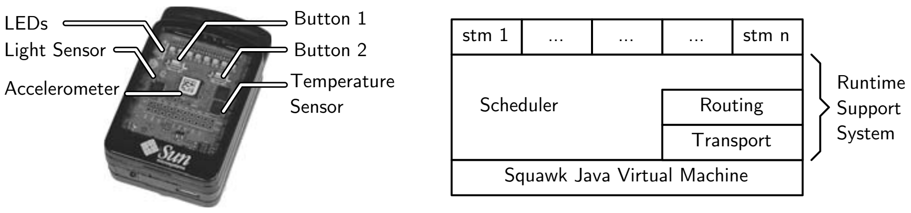


Fig. 1. Sun SPOT and Runtime Support System

1.2 Runtime Support System

To facilitate the execution of many concurrent processes on Sun SPOTs, we have implemented a runtime support system [6], sketched on the right side of Fig. 1. It includes a scheduler that is responsible for triggering the execution of state machine transitions whenever signals are received or timers expire. Further, a router and an object responsible for the transport of signals support communication using the SPOT's radio communication. For a detailed description of

[1] Partially funded by the Research Council of Norway, project #180122.

the execution mechanisms and their formal behavior in temporal logic, we refer to [7]. To generate the state machine classes from UML state machines, we use the code generator described in [8,9], which produces the necessary Java code.

1.3 The SPACE Engineering Method

We developed the method SPACE [1,2] for the engineering of reactive systems. This method focuses on the definition of reusable building blocks expressed as UML activities and collaborations, combined with Java code for detailed operations. Building blocks are grouped into libraries for specific domains, as illustrated on the left hand side of Fig. 2. Developers can use these blocks by composing them together within UML collaborations and activities: the collaborations describe the structural binding of roles and provide a high-level overview and activities describe the detailed behavioral composition of events, with some additional glue logic where necessary. Each block has an associated *external* state machine, abbreviated ESM, that provides a behavioral contract describing in which sequence parameters must be provided to or may be emitted by a block. This description is useful for understanding a block without looking at its internal details, and enables compositional model checking, as we describe below.

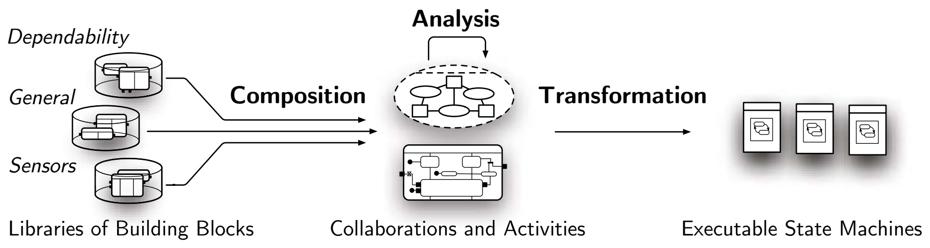

Analysis
Dependability
General
Sensors
Composition
Transformation
Libraries of Building Blocks
Collaborations and Activities
Executable State Machines

Fig. 2. The SPACE engineering method

Once a specification is complete, it is analyzed to ensure various properties that should hold for any application. For example, a composition of blocks should never harm any of the contracts (ESMs) and a collaboration should terminate consistently. For this behavioral analysis, we use model checking. Due to the compositional semantics and the encapsulation of building blocks by their ESMs, the state space needed for model checking tends to be very small, since only one building block on a single decomposition level has to be considered at a time.[2]

Complete systems are represented by special system collaborations and activities. When a system is sound, it can be transformed automatically into executable state machines and components, using a model transformation [10,11]. From the resulting state machines, code for different platforms (such as the Sun SPOTs introduced above) can be generated.

[2] We observe that most building blocks in our libraries require far less than 100 states.

2 A Sensor Network for Remote Home Monitoring

An increasingly popular area for home automation is to remotely monitor vacation homes and cabins. Several sensors can be installed in a cabin. One of the assumptions in our project is that embedded sensors with processing capacity similar to Sun SPOTs are so cheap they can also be used in a consumer market. For instance, the sensors can register the temperature at several places, detecting frost or fire. Further, they can detect sudden changes in light or measure acceleration on doors and windows, indicating that somebody is breaking in. With the extension card presented in [12], we further assume that each Sun SPOT is capable of GSM communication to set off an alarm to a remote user, for example by means of an SMS.

To improve the quality and robustness of the system, the sensors communicate among each other before sending an alarm via GSM. This serves several purposes: First, multiple sensors can be used redundantly, so that important conditions are monitored by more than one sensor, whereas only one alarm should be issued. Second, some conditions may give rise to alarm if the sensors are triggered in a certain pattern. For example, while changes in light of one sensor could indicate a broken window shutter, a change observed by several sensors may simply be due to a cloud moving in front of the sun.[3] This means that alarms need to be coordinated. For that reason, we use a leader election protocol that points out one SPOT sensor to filter and issue alarms. If the leader runs out of battery or otherwise fails, a new leader takes over. Such a network is illustrated in Fig. 3.

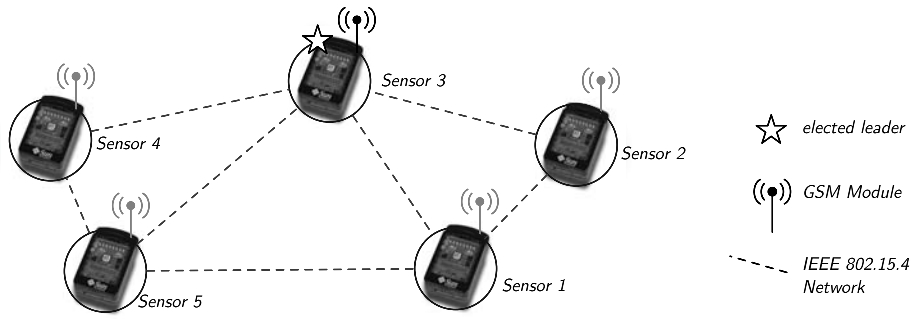


Fig. 3. Sensor network with the elected leader

Figure 4 shows the UML activity describing the behavior of a SPOT sensor as composed from our reusable building blocks. Since the SPOT sensors of the system all have the same behavior, it suffices to specify only one of them. To visualize the relationship of a SPOT sensor to the other sensors explicitly, however, we use two activity partitions. The left one, *spot sensor*, describes how a SPOT sensor is composed from building blocks, which defines the behavior. The

[3] We will not discuss detailed patterns describing when an alarm should be triggered, and we will also disregard the configuration of individual SPOT sensors.

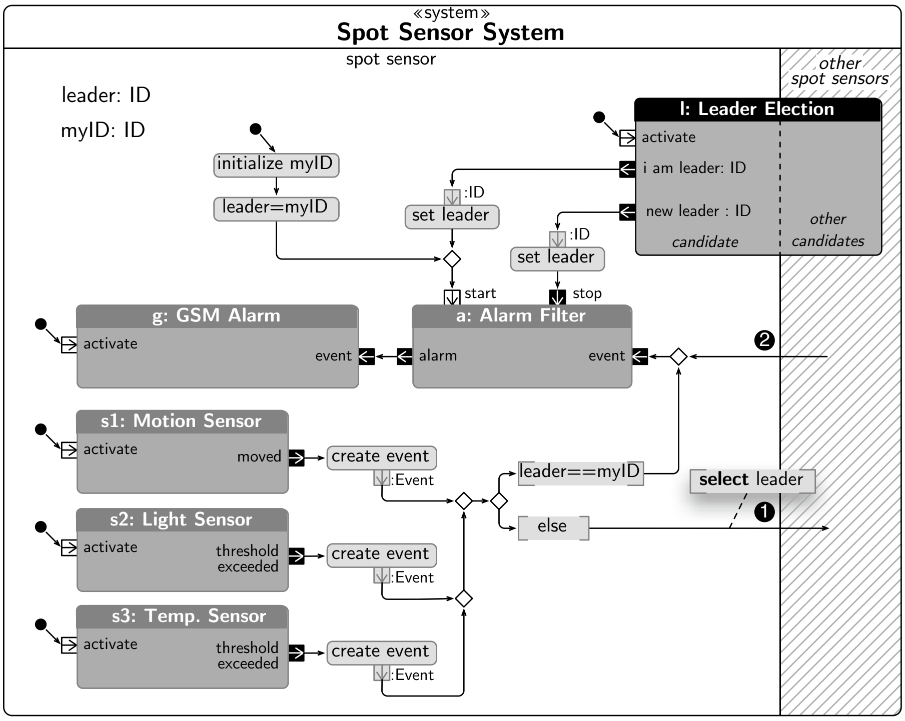


Fig. 4. Activity describing the composition of SPOT sensors from building blocks

right partition, *other spot sensors*, enables us to represent the communication with the other sensors. This partition is only sketched, as only the left one will be used for the transformation and code generation.

A sensor consists of a block[4] for GSM communication g, the alarm filter a and three building blocks accessing the Sun SPOT's sensors for motion ($s1$) light ($s2$) and temperature ($s3$). While these blocks encapsulate local behavior, a building block can also comprise collaborative behavior that is executed by several participants. The leader election, contributed by building block l in Fig. 4, is a typical example for that. It is a collaboration among several SPOT sensors, and therefore crosses the activity partitions. Internally, the block specifies the establishment of contact between all the sensors and how a leader is selected amongst them. This behavior is detailed in Sect. 4.

The activity also contains references to the operation *create event*. Since UML does not have a concrete language for actions, the details of these operations are specified by Java methods, managed by our editor. The other elements in the activity are initial nodes (●) as well as merge and decision nodes (◇). Decision nodes are followed by flows that are guarded ([ ]).

4 Technically, blocks are modeled as UML elements of type *Call Behavior Action*, which can refer to subordinate activities.

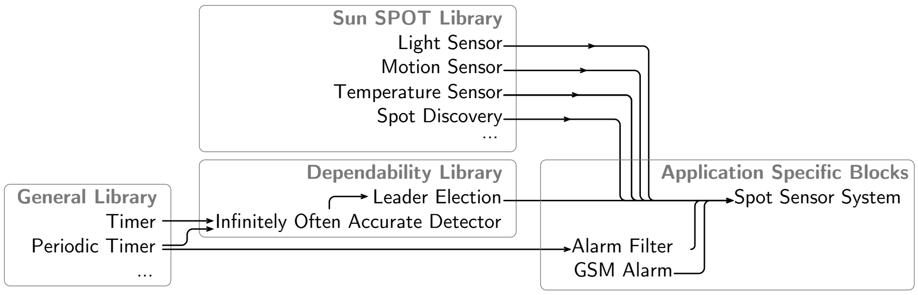


Fig. 5. Overview of reused block from libraries and application-specific blocks

Upon the start of a SPOT sensor, the initial nodes emit a token and start all blocks, including the collaboration for the leader election. The alarm filter is started as well, so that the SPOT by default uses its own GSM Alarm block to send any SMS notifications, until it finds another leader. The leader election emits a token through *new leader* once it detects a SPOT that is pointed out as the new leader, carrying its ID. In case a SPOT itself is pointed out as leader, a token is emitted through *i am leader*. In both cases, the ID of the leader is stored in variable *leader*. If a SPOT becomes leader, the alarm filter is started, and if the SPOT loses its leader status, the alarm filter is terminated.

Whenever one of the sensors *s1*, *s2* or *s3* registers a condition, it emits a token via its output pin, upon which an event is created containing the kind of condition and ID of the sensor. If the SPOT owning the sensors has the leader role (i.e., guard *leader==myID* is valid), the event is directly passed to the alarm filter. Otherwise, the SPOT sensor forwards the event to the current leader. In this case, the leader is one of the other SPOT sensors, and sending to it is specified by the transfer edge ❶. Since the other SPOTs are potentially many, we have to select which one to address, using the **select** operator introduced in [10]. It refers to the ID of the leader. Vice versa, if a SPOT sensor has the leader role, it may receive events from other SPOT sensors (at ❷).

Figure 5 provides an overview of the dependencies between the building blocks used for the specification of the SPOT sensor system. Most of them are taken from our existing libraries (listed here with only those blocks used in the example). The Alarm Filter, the experimental GSM Alarm, and the complete system are specific for the example.

3 Building Blocks Specific for Sun SPOTs

Our library for Sun SPOTs contains twelve building blocks dedicated to the specific capabilities of the devices, such as the buttons, all sensors on the SPOTs, and the LEDs. In the following we present some of those that are used in the SPOT sensor system.

3.1 Building Block for Sensors

Figure 6 shows the internal details of the block for the detection of movements. The accelerometers of the Sun SPOTs are accessible via a special API. To react on sudden accelerations that exceed a certain threshold value, a listener is registered at the SPOT classes that provide access to the hardware. To keep the execution of the code reacting upon an event under the control of the scheduler of our runtime support system (RTS), the building block uses an internal signal as buffer, to decouple the processes. For this reason, operation *register listener* creates a listener, which, upon its invocation following a sudden movement, produces a signal *MOVED*, that is fed into the RTS. Once this signal is processed, the behavior following the accept signal action declared for *MOVED* in Fig. 6 is executed: a token is emitted via output node *moved*, and the listener is reactivated, to listen for further movements. The blocks controlling the light and temperature sensors access the SPOT API in a similar way.

On the right hand side of Fig. 6, the ESM for the motion sensor is shown. As mentioned previously, it documents the behavior visible at the pins of an activity, so that we know its external behavior when it is instantiated as a block as in Fig. 4. Due to the ESM, we know that after a token enters *activate*, tokens may be emitted via *moved* until we terminate the block via *stop*.

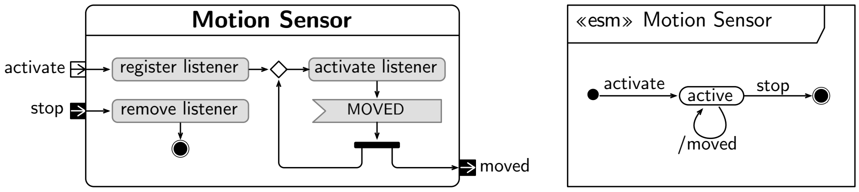


Fig. 6. Building block for the motion sensor

3.2 SPOT Discovery

To dynamically find other SPOTs in the sensor network, we provide a collaborative building block which uses the Sun SPOT's broadcasting functions so that they can discover each other. The corresponding activity is shown in Fig. 7. The partition *beacon* describes how a SPOT that wants to be discovered sends out periodic messages. Since these messages are specific for Sun SPOTS, they are sent directly from the Java operation, instead of using our runtime support system. The partition *listener* describes the logic to be implemented by a Sun SPOT that wants to discover other SPOTs. For that, it listens to the incoming beacon messages. To decouple the receiving processes from the scheduling of state machine transitions, once such a message arrives, it is fed into our RTS via signal *FOUND*, similar to the listener reacting to the movement of a SPOT explained above. If the ID is not yet known, a token is emitted via *found spot*. Notice that if a SPOT wants to both discover other SPOTS and be discovered, it instantiates this collaboration twice, once as a beacon and once as a listener.

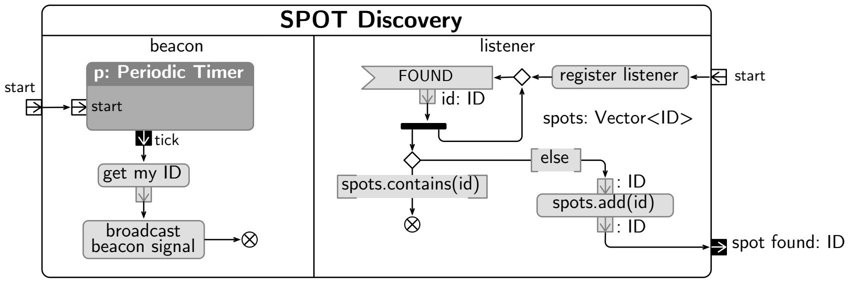


Fig. 7. Building block for the service discovery

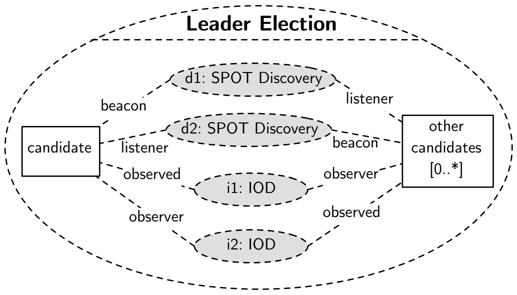


Fig. 8. Collaboration for the leader election

4 Collaborative Building Blocks for Leader Election

To make sure that only one of the SPOT sensors forwards an alarm over GSM, we use a fault-tolerant leader election protocol. Should the leader SPOT run out of battery or otherwise fail, another one must take its place so that alarms are still sent if necessary. To solve this problem, we implemented an algorithm from [13]. The algorithm uses an *Infinitely Often Accurate Detector* (IOD) as failure detector, a concept from [14], which is used by a component to monitor if any of its communication partners have crashed.[5] In Sect. 4.1 we provide a dedicated building block for this function.

The collaboration in Fig. 8 specifies the structural aspects of the leader election. It depicts the participant *candidate* as collaboration role, and refers to the sub-services for SPOT discovery and failure detection by collaboration uses *d1, d2* and *i1, i2*. The leader election is a symmetric collaboration, in which all participating roles have the same behavior, and the role for the candidate is therefore represented twice. For the model transformation and the code generation, the left *candidate* is used. To make the collaboration with the other candidates explicit, we refer to the *other candidates* on the right hand side, similar to our proceedings with the SPOT sensors in Sect. 2.

[5] In the fault-tolerance domain, a node is said to *crash* if it from some point on permanently ceases all operations, but works correctly until then (see [15]).

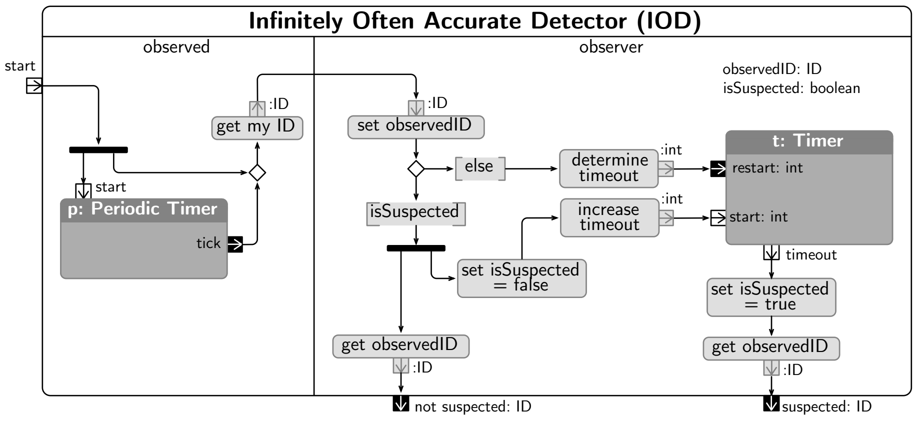


Fig. 9. Building block for the Infinitely Often Accurate Detector

4.1 Infinitely Often Accurate Detector (IOD)

In our example, we use the *Infinitely Often Accurate Detector* (IOD, [13]) as specified in Fig. 9. The partition on the left side models the observed SPOT, which periodically sends so-called "alive" messages to the observing SPOT, represented by partition *observer*. These messages are triggered by the periodic timer p and carry the ID of the observed SPOT. The observer SPOT maintains two variables to store the status of the observed SPOT; *observedID* for its ID and the boolean *isSuspected*. Moreover, the observer has a timer t to determine if the alive message from the observed SPOT is delayed.

Whenever the observer receives an alive message from the observed SPOT, it reacts depending on the current value of *isSuspected*:

– If the observer does not suspect the observed SPOT sensor of having crashed, it will simply restart timer t and wait for the next alive message.
– If, on the other hand, the observer currently suspects the observed SPOT of having crashed, the observer will change *isSuspected*, increment the timeout period[6] and emit the observed's ID through output node *not suspected*.

If, however, timer t expires (i.e., no alive message was received in time), the observer will suspect the observed SPOT of having crashed, set *isSuspected* accordingly and emit a token carrying the observed SPOT's ID through output node *suspected*.

Since a message could also be delayed in the communication medium, a timeout does not always mean that a SPOT has crashed. Hence there may exist transient states in which two SPOTs are both considered the leader. This, however, is acceptable for our application domain. For a detailed analysis and proof of the properties of the Infinitely Often Accurate Detector, we refer to [13].

[6] Incrementing the timeout period upon detecting a false suspicion ensures that the observer will wrongly suspect the observed only a limited number of times.

4.2 Composed Building Block for the Leader Election

The detailed behavior of the leader election is expressed by the activity in Fig. 10. Similar to the overall system of Fig. 4, the leader election is symmetric. The partition *candidate* on the left side represents one participant and its detailed behavior, while the partition to the right represents its communication partners.

As part of the leader election, a SPOT participates in the Infinitely Often Accurate Detector (IOD) collaboration as both *observer* and *observed* entity. This is represented by blocks *i1* and *i2*, which both refer to the activity in Fig. 9, but which are bound to partition *candidate* with roles *observed* resp. *observer*. Moreover, this collaboration is executed as multiple concurrent sessions (once towards each communication partner). This is signified by the shadow around them, a notation introduced in [10].

When the leader election collaboration is activated, the *SPOT Discovery* collaboration is initialized as both beacon (*d1*) and listener (*d2*), according to the role binding in Fig. 8, so that a SPOT sensor can both detect others and be detected by others. For each sensor found, a token with its ID is emitted via pin *spot found* of *d2*. This ID is used to start a new session of the IOD collaboration *i1*, so that a SPOT is observed by any other SPOT it detects. For that we use again the **select** statement, which this time refers to the value provided by the token flow. Vice versa, once a SPOT is detected by other SPOTS, they start a new instance of the IOD collaboration (in this direction represented by *i2*).

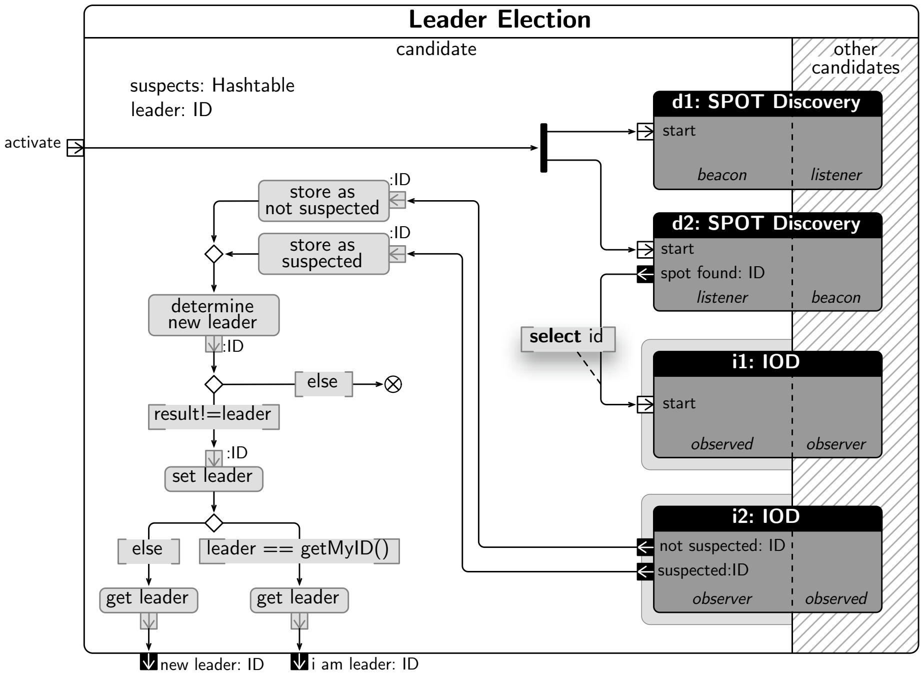


Fig. 10. Building block for leader election

Via the output pins *suspected* and *not suspected* on *i2*, a SPOT is notified about perceived changes in the state of each of the other SPOTs. The logic that follows determines the current leader status. For that, hash table *suspects* maps the ID of the other SPOTs to their respective status (suspected or not suspected). Whenever *i2* issues a change in state of another SPOT via one of its output pins, the subsequent operations store this change to the hash table and determine the new leader. If several SPOTs qualify for the leader status, the one with the lowest ID is chosen. If the leader has changed, we store the new leader and check if the new leader is this SPOT. Depending on the outcome, a token is emitted through either the *i am leader* or *new leader* output node.

5 Automated Analysis

The analysis of the specification is based on model checking. This process is automated, since our tool also generates the corresponding theorems to be verified. Currently, we check the following generally desirable system properties [16]:

- A building block must conform with its own ESM. The motion sensor of Fig. 6, for instance, may not emit a token via node *moved* after the surrounding context provided one via *stop*.
- A building block must also obey all ESMs of the subordinate blocks it is composed from.
- Building blocks with more than one participant are checked for bounded communication queues. For the IO detector in Fig. 9, for instance, we find that the periodic timer could, in principle, overflow the queue between the observing and the observed component.[7]

The analysis focuses on the soundness of interactions among collaboration participants as well as the correct composition of all building blocks with respect to event orderings. The content of operations (that is, the Java code) is not part of the analysis. In cases where decisions are involved that depend on variables, the analysis always examines all alternative branches. If the executions of some branches may harm certain properties, we reason manually if these cases may in fact happen. For instance, in the IO detector of Fig. 9, the else branch may restart the timer before it is started. This, however, never happens in the final system because of the value of *isSuspected*.

The results of the analysis are presented to the user by explanatory annotations within the original UML model, so that no expertise in the underlying formalism is required, as demonstrated in [17]. In addition, counter examples illustrating design flaws are presented as animations within the activities. In our experience, checking the above mentioned properties is of great value in the practical development of specifications. Although these properties may appear simple when considered in isolation, even experienced engineers usually harm

[7] In this case, however, we estimate the time needed for the transmission and subsequent processing and conclude that this is not an issue in a real system.

several of them in initial designs, especially when more complex collaborations are constructed.

Due to the compositional semantics of our method, each building block can be analyzed separately. Internal building blocks are abstracted by their ESMs, so that the global state space of the specification in Fig. 4 has only 15 distinct reachable states. Moreover, since most of the building blocks are taken from libraries and are already analyzed, only the new ones created for the specific applications have to be examined. These are the ones for the *SPOT Sensor System*, the *Alarm Filter* and the *GSM Alarm*.

6 Automated Implementation

As briefly mentioned in the introduction, the implementation is performed by a completely automated process with two steps: In a first step, executable state machines are synthesized from the activities. In a second step, code is generated. This is possible since the activities provide descriptions that are behaviorally complete, and the details of operations are provided as Java methods as part of the building blocks.

6.1 Transformation to Executable State Machines

In Fig. 11 and 12, we present the state machines as generated by the transformation. In our method, they are only an intermediate result used as input for the subsequent code generation; developers do not have to edit or read them. In the following, we highlight some properties to demonstrate the soundness of the transformation.

For the partitioning of components into state machines (or *processes* in SDL), our algorithm follows the guidelines from [6]. In particular, the algorithm merges all behavior of building blocks that is executed one at a time by the component under construction into one single state machine. All blocks that denote multi-session collaborations (behavior that is executed multiple times towards a changing number of different communication partners) are implemented by dedicated state machines, one instance for each session, as presented in [10]. For the SPOT sensor system, for instance, the algorithm creates the state machine *Spot Sensor*, depicted in Fig. 11, which takes care of the main component behavior. This includes all logic contained in the building blocks used in Fig. 4. However, since the behavior of the Infinite Often Accurate Detector is executed concurrently within each SPOT sensor (once for each other sensor detected), its behavior is implemented by dedicated state machines. These are state machines *Observer* and *Observed* in Fig. 12.

The main state machine *Spot Sensor* has two[8] distinct control states, *1* and *2*. This is because the transition behavior only has to distinguish if a SPOT is

[8] This is less than the 15 states from the previous analysis because the analysis also captures the interleaving with other SPOTs and the queues for communication, which do not contribute any control states for a local component.

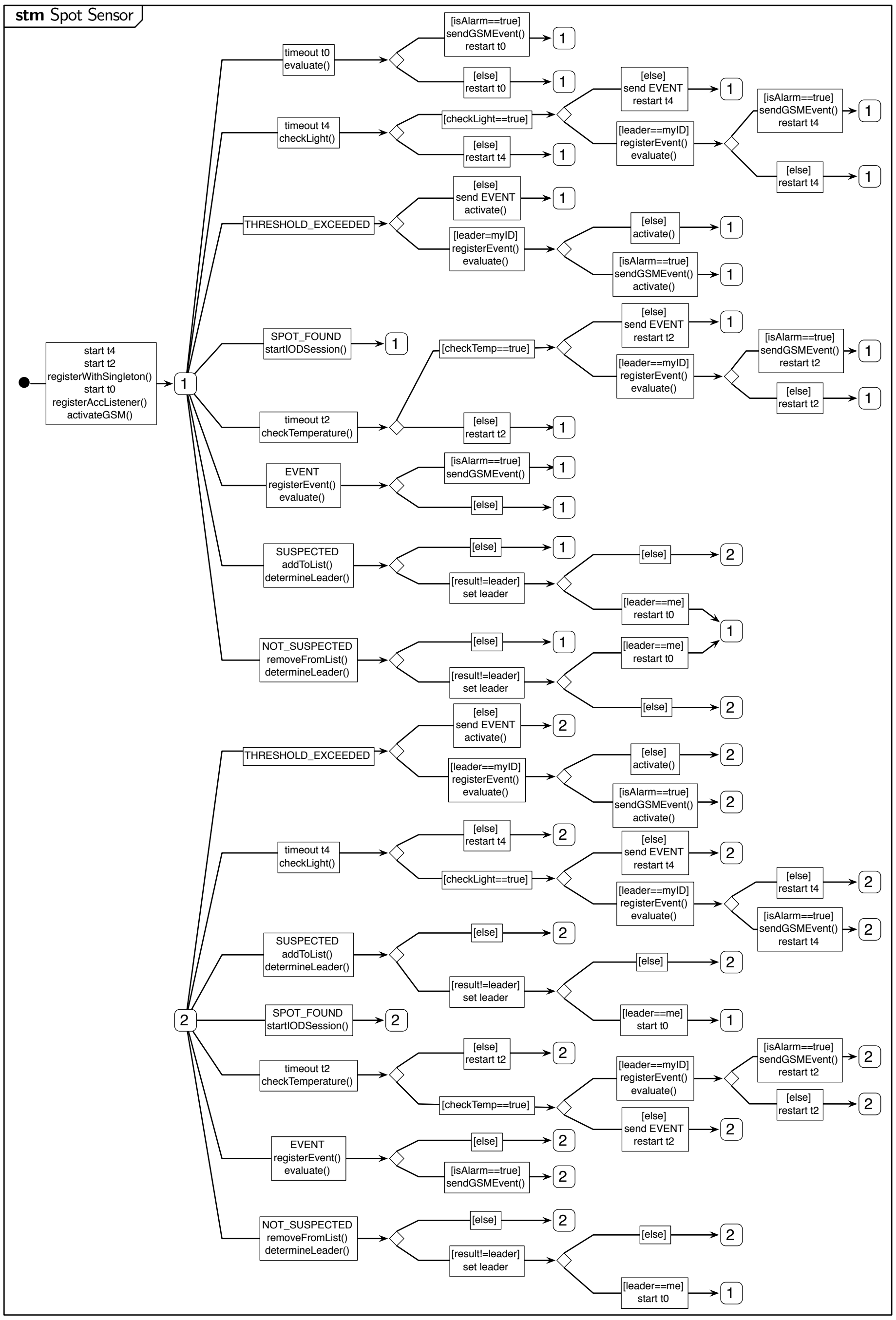


Fig. 11. Bird's eye view of the synthesized state machine for the Spot Sensor

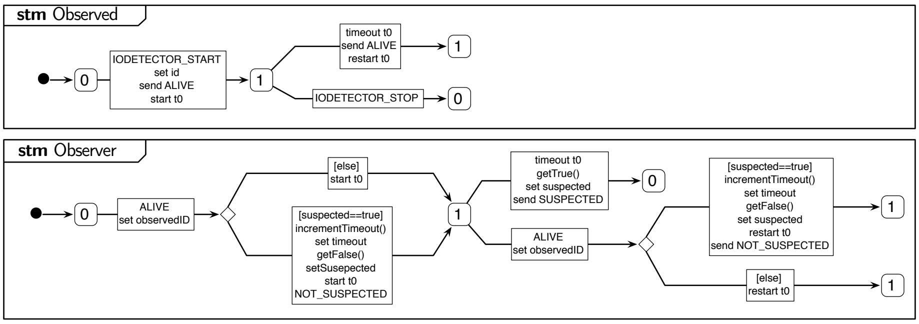


Fig. 12. The synthesized state machines for the IO detector

the leader or not. When a spot is the leader, the alarm filter is active and the state machine is in state *1*. When another SPOT is the leader, the alarm filter is inactive and the state machine is in state *2*. The transitions from either state handle the periodic checks of the sensors, the periodic discovery protocol and react to the events of the Infinitely Often Accurate Detector. In state *1*, which is entered by the initial transition, the SPOT assumes it is the leader and therefore starts the alarm filter, which constantly evaluates the log of events, shown by the topmost transition.

6.2 Code Generation for Sun SPOTs

Since the Sun SPOTs execute Java, the code generator described in [9] is largely based on the standard Java code generator, described in [18]. As introduced in Sect. 1.2, the execution is based on a runtime support system, which takes care of scheduling, routing and transport of messages. The scheduler (see Fig. 1) maintains event queues for each state machine in which incoming messages and active timers are placed. In a round-robin manner, the scheduler triggers the execution of state machine transitions by feeding the event into a dedicated transition method, which is specific for each state machine type. The transition method contains nested if-statements that distinguish the current control state and input event and then execute the effect as specified by the UML transitions in Fig. 11 and 12. Effects referring to operation calls on the activity level, such as *determine new leader* in Fig. 10, are copied into the transition method. Other actions that are part of a transition effect, such as sending signals or operations on timers, are synthesized from the UML model. The transport module (see Fig. 1), responsible for sending and receiving messages from and to other SPOTs, uses the the radio stream protocol from the Sun SPOT API to transmit messages. This protocol provides buffered, reliable, stream-based communications over multiple hops on top of the IEEE 802.15.4 radio protocol. The content of the messages sent via the radio channels are SOAP-documents generated with the help of the kSOAP libraries [19], as described in [8]. For the necessary serialization of objects, the code generator adds methods that convert objects and primitive types to strings.

7 Estimation of Reuse Proportions

To estimate the degree of reuse for the exemplified system, we distinguish between the building blocks that are part of our libraries and intended for reuse, and those building blocks constructed specifically for the application. These are shown in Fig. 5, with the libraries on the left hand side. As application-specific we count the *Alarm Filter*, the *GSM Alarm* and the overall *SPOT sensor* system. The effort necessary for the construction of a building block consists of the UML models on the one hand and Java code contained within the call operation actions (like *determine new leader* in Fig. 10) on the other hand.

– By counting the lines of code contained in the call operation actions in each building block, we find that there are $l_{blocks} = 443$ lines of code within the call operation actions for all building blocks used in the system in total. Those building blocks taken from libraries contribute with $l_{lib} = 333$ lines, so that the reuse proportion $R_{code} = l_{lib}/l_{blocks}$ is 75 %.
– As an estimate for the effort spent UML modeling, we use a simple metric that just counts the number of activity nodes and activity edges $n = n_{nodes} + n_{edges}$ within a building block. This metric shows that all building blocks used in the system consist of $n = 276$ edges and nodes in total. Those building blocks taken from the library contribute with $n_{lib} = 195$ elements, so that the reuse proportion $R_{model} = (n_{lib}/n)$ is 71 %.

Of course, these numbers vary for different systems. For the given example, we have programmed a relatively simple logic for the alarm filter, which contributes only 50 lines of code. Since the GSM module is not yet finalized, we estimate another 50 lines for that building block.

To get an impression of the overall gains including the automatic implementation, we consider also the complete code needed for the execution on top of the runtime support system. The code generated automatically for the state machine logic adds up to $l_{stm} = 634$ lines, and the number of code lines written manually for the Java operations copied from the building blocks as mentioned above is $l_{blocks} = 443$. This means that the code necessary for the entire application has $l_{total} = l_{stm} + l_{blocks} = 1077$ lines,[9] from which $l_{stm}/l_{total} = 59$ % are generated automatically. If we add up these numbers, we find that $(l_{lib} + l_{stm})/l_{total} = 90$ % of the Java code lines are either reused or generated from the UML models.

8 Related Work

There exist a number of approaches for the model-based design of reactive systems that are also suitable for embedded applications. Some of them based on SDL such as TIMe [20], SPECS [21], SOMT [22] and SDL-MDD [23]. Others,

[9] The underlying runtime support system has about 1900 lines of code. Since it is provided as a library that can be reused also in manual approaches, it is not part of our calculation.

such as ROOM [24] (later UML-RT) or Catalysis [25], are oriented towards UML as language. As design models that describe the behavior of individual components, these approaches use state machines, either in the form of SDL processes or as UML state charts (called *ROOM charts* in [24]). To capture collaborative behavior among several components, most of these approaches rely on MSCs. Catalysis [25], inspired by the Object-Oriented Role Analysis Method (OOram, [26]) and DisCo [27], on the other hand, uses collaborations more explicitly in specific diagrams, albeit in a rather informal way that requires manual synchronization by the developers. Micro protocols [28] are another approach to capture and encapsulate communication protocols within self-contained units, by using pairs of SDL processes or composite states.

In principle, these approaches are compatible with the one presented here, since all the design models based on state machines with their emphasis on event-driven transitions are quite similar. The difference lies in the models on which developers work: To enable the composition of collaborative behavior as self-contained building blocks, we use UML activities, from which the state machine-based design models are derived automatically. This enables a number of opportunities for the reusability, the analysis and the overall specification style, as we will argue below.

9 Concluding Remarks

In our experience, the composition as enabled by activities, shown for example in Fig. 4, is quite flexible. We attribute this to two major reasons: First, the complete but cross-cutting nature of UML activities, in which the coordination of several participants can be described within the same diagram. If, for example, we would like to exchange the selected leader election protocol with another one, we would just have to replace the building block l in Fig. 4, and its connections to the other blocks, which can be achieved by focusing on one single diagram. Second, the way activities enable the encapsulation of functionality related to a certain purpose as separate, self-contained building blocks. While state machines offer some means of structuring (for example composite states), they do not offer the same degree of flexibility and separation as activities. The functions encapsulated by the building blocks in Fig. 4, for example, are dispersed among several transitions in the state machines of Fig. 11 and 12. One reason for that is that state machines represent their states by explicit control states, while activities use concurrent flows that may execute independently. Although such behavior can to a certain degree be described in state machines by concurrent regions, such a description style gets intricate once the behaviors in these regions need to be synchronized. However, since state machines are very suitable for the specification of the executable behavior of components, we generate them in the described way, so that we have both the compositional features of UML activities and the efficient scheduling of state machines.

Besides these properties coming from the chosen notation, an important feature of our method is the compositional verification it enables, based on the underlying semantics in cTLA [29]. Not only does this reduce the state space during model checking, but it also has important effects on the larger scale development process. Since building blocks can be verified individually, proven solutions can be encapsulated in building blocks, and these can be checked and stored in a library. Whenever a building block is reused, the verified properties are enforced automatically and do not have to be re-verified. This enables "true reuse" as mentioned in [25], in which reuse does not mean to simply copy and paste some parts of a specification, but also ensures that important properties are maintained.

All things considered, we think that the chosen principles and the way they are combined enable a reuse-oriented specification style, one that encourages the use of encapsulated building blocks to a high degree, but that still allows us to adapt systems to match the requirements of the individual application. This is a crucial step towards the cost-effective LEGO-brick like development paradigm.

Acknowledgements. We would like to thank Fritjof Boger Engelhardtsen from Telenor for support with the GSM module, as well as Marius Bjerke and Bemnet Tesfaye Merha for their enthusiastic work on the Sun SPOTs in their theses.

References

1. Kraemer, F.A.: Engineering Reactive Systems: A Compositional and Model-Driven Method Based on Collaborative Building Blocks. PhD thesis, Norwegian University of Science and Technology, Trondheim (August 2008)
2. Kraemer, F.A., Herrmann, P.: Service Specification by Composition of Collaborations — An Example. In: Proceedings of the 2006 WI-IAT Workshops (2006 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology), pp. 129–133. IEEE Computer Society Press, Los Alamitos (2006)
3. ISIS Project Website, http://www.isisproject.org
4. http://www.sunspotworld.com
5. http://squawk.dev.java.net/
6. Bræk, R., Haugen, Ø.: Engineering Real Time Systems: An Object-Oriented Methodology Using SDL. Prentice-Hall, Englewood Cliffs (1993)
7. Kraemer, F.A., Herrmann, P., Bræk, R.: Aligning UML 2.0 state machines and temporal logic for the efficient execution of services. In: Meersman, R., Tari, Z. (eds.) DOA 2006. LNCS, vol. 4276, pp. 1612–1632. Springer, Heidelberg (2006)
8. Bjerke, M.: Asynchronous Messaging between Embedded Java Devices. Project Thesis. Norwegian University of Science and Technology, Trondheim (December 2008)
9. Merha, B.T.: Code Generation for Executable State Machines on Embedded Java Devices. Project Thesis. Norwegian University of Science and Technology, Trondheim (December 2008)
10. Kraemer, F.A., Bræk, R., Herrmann, P.: Synthesizing Components with Sessions from Collaboration-Oriented Service Specifications. In: Gaudin, E., Najm, E., Reed, R. (eds.) SDL 2007. LNCS, vol. 4745, pp. 166–185. Springer, Heidelberg (2007)

11. Kraemer, F.A., Herrmann, P.: Transforming Collaborative Service Specifications into Efficiently Executable State Machines. In: Ehring, K., Giese, H. (eds.) Proceedings of the 6th International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2007). Electronic Communications of the EASST, vol. 7. EASST (2007)
12. Arseneau, E., Engelhardtsen, F.B.: Project playSIM: Experimenting with Java Card 3 System Programming. In: JavaOne (June 2009)
13. Garg, V.K.: Elements of Distributed Computing. John Wiley & Sons, Inc., New York (2002)
14. Chandra, T.D., Toueg, S.: Unreliable Failure Detectors for Reliable Distributed Systems. Journal of the ACM 43(2), 225–267 (1996)
15. Tanenbaum, A.S.: Distributed Systems: Principles and Paradigms. Prentice-Hall, New Jersey (2002)
16. Kraemer, F.A., Slåtten, V., Herrmann, P.: Engineering Support for UML Activities by Automated Model-Checking — An Example. In: Proceedings of the 4th International Workshop on Rapid Integration of Software Engineering Techniques, RISE (2007)
17. Kraemer, F.A., Bræk, R., Herrmann, P.: Compositional Service Engineering with Arctis. Telektronikk 105(1) (2009)
18. Kraemer, F.A.: Rapid Service Development for Service Frame. Master's thesis, University of Stuttgart (2003)
19. http://ksoap2.sourceforge.net/
20. Bræk, R., Gorman, J., Haugen, Ø., Melby, G., Møller-Pedersen, B., Sanders, R.: Quality by Construction Exemplified by TIMe — The Integrated Methodology. Telektronikk 95(1), 73–82 (1997)
21. Olsen, A., Færgemand, O., Møller-Pedersen, B., Reed, R., Smith, J.R.W.: Systems Engineering Using SDL-92, Chapter 6 – Systems Engineering. Elsevier North-Holland, Inc., Amsterdam (1994)
22. Telelogic: Tau 4.4 User's Manual. Malmö (2002)
23. Kuhn, T., Gotzhein, R., Webel, C.: Model-Driven Development with SDL - Process, Tools, and Experiences. In: Nierstrasz, O., Whittle, J., Harel, D., Reggio, G. (eds.) MoDELS 2006. LNCS, vol. 4199, pp. 83–97. Springer, Heidelberg (2006)
24. Selic, B., Gullekson, G., Ward, P.T.: Real-Time Object-Oriented Modeling. John Wiley & Sons, Inc., New York (1994)
25. D'Souza, D.F., Wills, A.C.: Objects, Components, and Frameworks with UML: the Catalysis Approach. Addison-Wesley, Reading (1999)
26. Reenskaug, T., Wold, P., Lehne, O.A.: Working with Objects, The OOram Software Engineering Method. Prentice-Hall, Englewood Cliffs (1995)
27. Jarvinen, H., Kurki-Suonio, R., Sakkinen, M., Systa, K.: Object-Oriented Specification of Reactive Systems. In: Proceedings of the 12th International Conference on Software Engineering, pp. 63–71. IEEE Computer Society Press, Los Alamitos (1990)
28. Fliege, I., Gotzhein, R.: Automated generation of micro protocol descriptions from SDL design specifications. In: Gaudin, E., Najm, E., Reed, R. (eds.) SDL 2007. LNCS, vol. 4745, pp. 150–165. Springer, Heidelberg (2007)
29. Herrmann, P., Krumm, H.: A Framework for Modeling Transfer Protocols. Computer Networks 34(2), 317–337 (2000)

Energy-Aware System Design with SDL[⋆]

Reinhard Gotzhein[1], Marc Krämer[1], Lothar Litz[2], and Alain Chamaken[2]

[1] Networked Systems Group
University of Kaiserslautern, Germany
{gotzhein,kraemer}@cs.uni-kl.de
[2] Institute of Automatic Control
University of Kaiserslautern, Germany
{litz,chamaken}@eit.uni-kl.de

Abstract. Energy consumption is a major concern during the development of wireless networks consisting of battery-powered nodes. In this paper, we study possibilities of specifying energy aspects in the system design phase, with SDL as design language. In particular, we strive for suitable abstractions, by establishing a design view that is largely platform-independent. We achieve this objective by identifying and realizing *energy mode signaling* and *energy scheduling* as two complementary approaches to incorporate energy aspects into SDL. We provide details on how to apply these approaches in order to control the energy consumption of CPU and transceiver, and how to implement them on the Imote2 platform. A case study illustrates the use of both approaches in a wireless networked control system.

1 Introduction

In systems consisting of battery-powered nodes, energy consumption is a major concern, constraining the uptime of nodes and therefore of networks. Among these systems are wireless sensor networks [1], ad-hoc networks [2], and networked control systems [3]. As a general rule, energy consumption should be minimized. This can, for instance, be achieved by choosing low-power hardware platforms, such as micro controllers and motes, and by devising energy-efficient algorithms and protocols. Also, hardware components can be temporarily shut off or switched to an energy saving mode when not needed. Such techniques are being applied with great success to laptop computers and mobile phones.

When minimizing energy consumption, it is important that the application requirements are taken into consideration from the beginning. For instance, in a networked control system, consisting of a set of sensors, actuators, and a controller communicating over a wireless medium, the stability of the controlled system is of primary concern. Energy consumption must only be reduced if the system can still be kept sufficiently stable. This calls for a cross-layer design approach, addressing

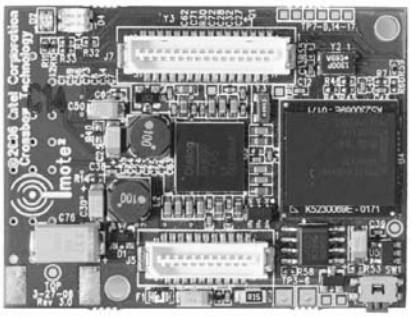
[⋆] This work is supported by the German Research Foundation (DFG) within the priority program SPP 1305 "Control Theory of Digitally Networked Dynamical Systems" under grants GO 503/8-1 and LI 724/15-1.

energy aspects already in the early development phases. In this paper, we focus on the design phase, referred to as *energy-aware system design*.

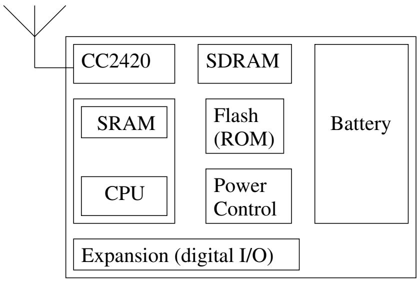
In the design phase, modeling techniques such as SDL [4] or UML [5] are often used. With these techniques, design models can be specified on an abstract level, relieving the developer from implementation-specific details. The design view would typically exclude resource aspects such as memory size, CPU speed, transmission rates, and energy consumption. However, if these aspects are a major concern and part of the system requirements, then suitable abstractions that can be exploited to optimize resource usage must be included in the design. For instance, in order to minimize energy consumption, it should be possible to identify periods of inactivity at execution time, based on the system design. Also, it should be possible to explicitly control the energy modes of individual hardware components on design level.

In this paper, we address energy-aware system design, with SDL as the design language. We examine the requirements of energy-aware design, and exploit existing language constructs of SDL for this purpose. Furthermore, we extend the implementation of the SDL virtual machine (SVM) in order to identify periods of inactivity and thereby reduce energy consumption. To further enhance energy-aware system design, we propose to exploit existing language constructs of SDL. Finally, we address implementation issues and provide a case study from the networked control systems domain to illustrate our findings. Our work uses the Imote2 platform [6] (see Fig. 1).

The paper is organized as follows: In Sect. 2, we survey related work. Section 3 introduces an energy model, which forms the basis for energy-aware design. In Sect. 4, we identify and realize two complementary approaches, called *energy mode signaling* and *energy scheduling*, to incorporate energy aspects into SDL. Section 5 shows how energy-aware design is implemented, by extending the SVM and SEnF, our SDL Environment Framework. Section 6 presents an excerpt of an inverted pendulum system, controlled over a wireless network. We draw conclusions in Sect. 7.



(a) Photo



(b) Block diagram

Fig. 1. The Imote2 hardware platform

2 Related Work

In the literature, a wealth of results on energy-aware systems and protocols has been reported. Practical system deployments are constrained by the available hardware components and energy sources. In [7], *battery models* for energy-aware system designs are considered. Benini et al. [8] give an overview of energy-aware designs of *embedded memories*. Current techniques to reduce the energy consumption of the CPU include *voltage scaling* [9]. An energy model of the *XScale platform* has been reported in [10].

To minimize the energy consumed by *transceivers*, a variety of energy-aware protocols have been devised. Among these are so-called duty cycling protocols on MAC level. Generally, the term *duty cycle* denotes the fraction of time a component (CPU, transceiver) or system is in an active state. If the traffic load of a network is below its capacity, it is possible to save energy by systematically switching transceivers to inactive mode. In the optimal case, a transceiver needs to be active only if it has to transmit or receive. The difficulty here is to determine this in advance. Several MAC protocols for duty cycling that attempt to approximate the optimal case have been reported, including S-MAC [11] and RMAC [12]. Similar considerations apply to routing in ad-hoc networks [13].

The *formal specification* of energy aspects as part of the system design has received little attention so far. Mohsen et al. [14] present an integrated methodology and tool support for voltage scheduling and energy-aware co-synthesis of real-time systems. It is mentioned that SDL has been used as design language; however, no details are given in the paper.

It is certainly straightforward to specify the *functionality* of energy-aware duty-cycling or routing protocols in SDL. Compiling these designs using available tools (e.g. Cmicro [15], ConTraST [16]) may even yield runtime-efficient code. However, without additional measures, this does not exploit the energy saving mechanisms of embedded platforms. Also, energy aspects are currently not incorporated into existing SDL runtime systems controlling the execution of code generated from SDL, nor are they made explicit in the SDL design.

3 Energy Model

To assess and minimize the energy consumption of a specific hardware platform, an *energy model* is needed. The energy model determines, for each hardware component, modes of operation, the energy consumed in each mode, and possible transitions between modes. Some of these transitions are triggered by the hardware itself. For instance, in case of a power fault due to a discharged battery, operation may be reduced to a minimum by switching to a deep sleep state, or even by switching off the entire node. This is usually done in a controlled fashion, i.e. outstanding memory transactions are completed, and volatile memory is placed in self-refresh mode or is saved to persistent memory. Other transitions between modes of operation may be accessible to the system programmer by setting a special power mode register, which provides the basis for the implementation of energy scheduling strategies. Based on the energy model, strategies for saving energy can be developed and evaluated. For instance, a CPU may be switched to idle mode or sleep mode during inactive periods. Also, a transceiver may be switched off between active phases of a duty cycle. As a general rule, the mode consuming the least energy should be chosen when components are temporarily not used. However, when changing a mode, the duration and power consumption to return to full activity must be considered. For instance, to switch from sleep mode to active mode may take considerably longer than from idle mode to active mode, as more hardware units have to be powered on. Which mode to choose therefore depends on the required reaction time and the total energy balance.

Figure 2 shows an excerpt of our energy model for the Imote2 platform: the upper part shows the CPU model; the lower part shows the transceiver model. The modes and transitions of the CPU model are documented in the Intel PXA27x Processor Family Developer's Manual [17]. Values for energy consumption are taken from [18]. The CC2420 data sheet [19] is used for the transceiver.

The CPU energy model is decomposed into two transition systems, representing mode of operation and CPU speed (frequency). In total, 6 modes of operation

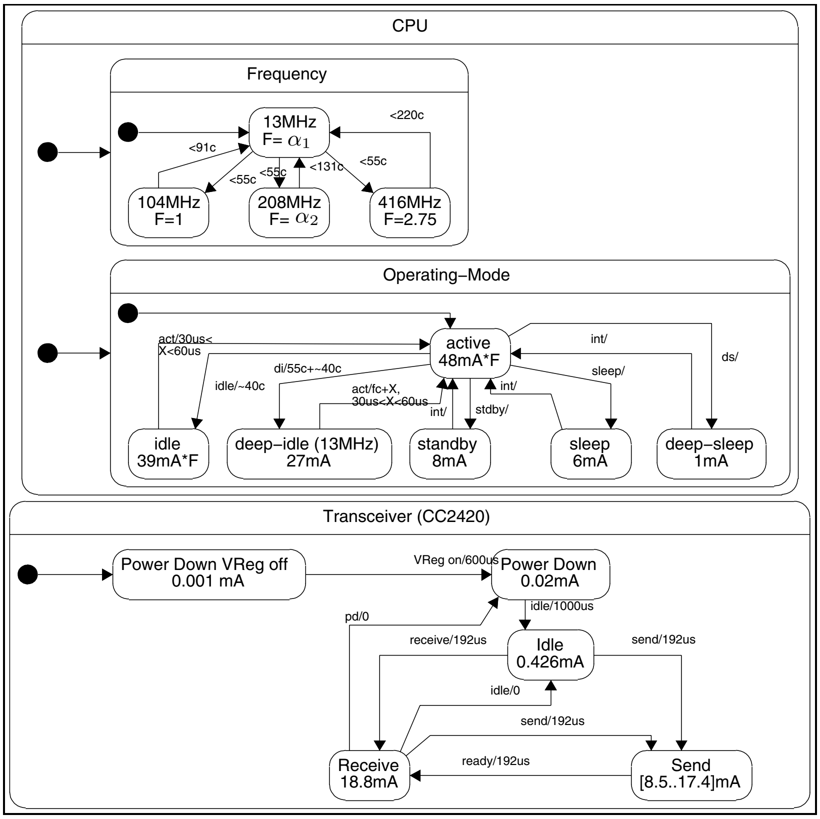


Fig. 2. Energy model of the Imote2

are distinguished, with power consumption ranging from 132 mA in active mode to 1 mA in deep sleep mode. In active and idle mode, power consumption also depends on the current CPU speed, which is expressed by the factor F. By frequency scheduling, the CPU speed may be adapted to the current task situation. There are 4 different speeds, ranging from 13 MHz up to 416 MHz. The (normal) speed of 104 MHz is the basis for energy calculations, therefore, $F = 1$. Transition durations are given in clock cycles. Note that some values are not reported in the available literature and therefore omitted.

The transceiver energy model shows 4 modes of operation and an intermediate mode during initialization. Interestingly, receiving consumes more energy than sending, which is typical for low power transmitters and must be considered by duty cycling protocols. Transition durations are given in s. When not needed, the transceiver may be switched to idle mode or even to power down mode, if the additional delay of $1000s$ to activate the transceiver can be granted.

4 Specifying Energy Control Aspects with SDL

In this section, we present our methodology for energy-aware system design with SDL. We assume the existence of an energy model, as discussed in Sect. 3. In Sect. 4.1, we introduce two conceptual approaches to incorporate energy aspects into SDL. We then apply these approaches to reduce energy consumption of CPU and transceiver in Sect. 4.2 and Sect. 4.3, respectively. As it turns out, this can be achieved in an intuitive, conceptually clean way on a level of abstraction that is entirely adequate for SDL designs.

4.1 Energy Mode Signaling and Energy Scheduling

To incorporate energy aspects into SDL, we have identified two complementary conceptual approaches. The first approach, called *energy mode signaling*, requires that the system developer specifies transitions of the energy model as part of the SDL design. In other words, transitions between modes of operation and thus the energy consumption of individual hardware components are controlled *explicitly*. This can be done, for instance, by sending special SDL signals to the environment, or by using special SDL procedures. In both cases, a specific treatment during system implementation is required to map energy mode signaling to the native operating system, and finally to the hardware platform. This treatment is different from the exchange of regular SDL signals with the environment, which are addressed to other processes of the local node or remote nodes. To distinguish between these cases, we apply naming conventions.

The second approach, called *energy scheduling*, incorporates additional functionality into the implementation of the *SDL Virtual Machine* (*SVM*) [20]. The SVM is the core of the SDL formal semantics [21] and provides typical operating system functionality on top of a logical hardware called *SDL Abstract Machine* (*SAM*). It controls the dynamic system architecture, the activity of SDL processes at runtime, the selection of transitions, and their firing. The additional functionality consists, for each hardware component, of an energy scheduler, which is responsible for triggering transitions of the energy model. The energy scheduler uses state information available to the SVM in order to determine when the mode of operation is to be changed. To derive this information, signal queues and active SDL timers are inspected. As this does not require modifications of the SDL design, energy scheduling is an *implicit* approach to controlling energy consumption.

Energy mode signaling and energy scheduling both have limitations. With energy mode signaling, the decision about changing the energy mode of a hardware component has to be based on information available to the SDL process executing the corresponding SDL action. In particular, this means that global conditions such as "currently no fireable transitions in the SDL system" cannot be used to switch the CPU to sleep mode. Furthermore, it means that the duration of the sleep period cannot be determined from the global set of active SDL timers. However, both kinds of information are available to the SVM, and can be used for energy scheduling, i.e. for implicit energy control. On the other hand, energy scheduling cannot determine, for instance, duty cycles of the transceiver from the current state of the SDL system. Duty cycles can be applied by using energy mode signaling to explicitly switch the transceiver between modes of operation based on SDL timers. In summary, we observe that energy mode signaling and energy scheduling are complementary and therefore should be applied together.

4.2 Controlling Energy Consumption of the CPU

In this section, we study how to apply energy mode signaling and energy scheduling in SDL in order to control the energy consumption of the CPU. Recall that *energy mode signaling* requires explicit action: the specification of SDL actions changing the mode of operation. For these actions to be executed, the CPU is required and therefore has to be in active mode (see Fig. 2). For the CPU, the following two cases can be considered:

CPU frequency signaling. Here, the designer adapts CPU speed to control energy consumption, thereby scaling system performance up or down. For this purpose, he defines a frequency scheduling strategy and uses energy mode signaling to switch between frequency modes. For instance, in case of the Imote2 energy model (see Fig. 2), switching between 13 MHz, 104 MHz, 208 MHz, and 416 MHz is possible. To signal frequency modes in SDL, we define a special SDL signal `CPU_FREQ_MODE`, which carries the frequency as parameter (see List. 1, Line 5). A specific frequency mode can then be signaled by sending `CPU_FREQ_MODE` to the environment (Line 7), where it is mapped to the native operating system.

CPU operation signaling. The designer switches the CPU from active mode to inactive mode for a specified period of time. For this purpose, he defines a duty cycling strategy for the CPU and uses energy mode signaling to switch to a

Listing 1. SDL excerpt: CPU frequency signaling in SDL

```
syntype CPU_Frequency = Integer
    constants 13, 104, 208, 416
endsyntype

signal CPU_FREQ_MODE (CPU_Frequency);

output CPU_FREQ_MODE (104);
```

particular inactive mode. For instance, in case of the Imote2 energy model (see Fig. 2), idle, standby, and sleep are inactive modes. Our SDL solution is analogous to CPU frequency signaling, and is shown in List. 4.2. Note that after the signal `CPU_OP_MODE` is sent (Line 7), the execution of the current SDL transition is suspended. It will be resumed after the CPU is switched back to active mode, triggered by a hardware timer expiring after the specified duration (since SDL timers are not available in inactive mode).

Listing 2. SDL excerpt: CPU operation signaling in SDL

```
1 newtype CPU_Operation
2   literals active, idle, deepIdle, standby, sleep, deepSleep;
3 endnewtype
4
5 signal CPU_OP_MODE (CPU_Operation, Duration);
6
7 output CPU_OP_MODE (idle, 0.01);
```

Next, we consider *energy scheduling*, which is achieved by incorporating additional functionality into the implementation of the SDL Virtual Machine (SVM) (cf. Sect. 4.1). For this, a scheduling strategy that is based on state information available to the SVM is to be devised. For the exposition, we assume that in the SVM implementation, transition selection for all agents of a node is done in a centralized manner, and that there are separate queues `signalQueue` and `timerQueue` for SDL signals and active SDL timers, respectively.

Listing 3 shows the CPU energy scheduler, written in pseudo code. It uses the SDL transition scheduler `selectTransition`, the transition handler `fireTransition`, and the timer routine `selectNextExpiringTimer`. If no transition is currently enabled, the scheduler searches the timer queue for the next expiring timer. If an active timer is found, the CPU is switched to sleep mode until either the timer expires, or an external interrupt occurs. Otherwise, the sleep time is eventually terminated by an external interrupt, which may e.g. be triggered by the environment of the SDL system when the transceiver receives a message.

It should be noted that the use of the SDL energy scheduler requires certain stylistic restrictions. First, we rule out the use of `now` in continuous signals, such as in the condition `(t>now)`. Instead, SDL timers should be used in this case. Second, we either disallow spontaneous transitions, or assume that they are never enabled when the CPU is in inactive mode.

Listing 3. CPU energy scheduler

```
while (running) do
    selectTransition
    if (transitionFound) then
        fireTransition
    else
        selectNextExpiringTimer in timerQueue
        if (timerFound) then
            sleep until (timer expires or interrupt occurs)
        else
            sleep until (interrupt occurs)
        fi
    fi
od
```

4.3 Controlling Energy Consumption of the Transceiver

In this section, we study how to apply energy mode signaling and energy scheduling in SDL in order to control the energy consumption of the transceiver. As *energy mode signaling* requires explicitly specified SDL actions to be executed at runtime, the CPU has to be in active mode. Since the energy model of the CC2420 transceiver has only modes of operation (see Fig. 2), it suffices to support *transceiver operation signaling.* Based on some duty cycling strategy, the transceiver is switched between active and inactive modes. For this purpose, we introduce a special SDL signal `CC2420_OP_MODE` (see List. 4), which carries the mode of operation as parameter. A specific mode of operation can then be signaled by sending `CC2420_OP_MODE` to the environment (Line 7), where it is mapped to the transceiver hardware. Note that the send mode is only signaled implicitly, i.e. when a frame is to be transmitted, and therefore omitted in the type definition `CC2420_Operation`.

Listing 4. SDL excerpt: transceiver operation signaling in SDL

```
1 newtype CC2420_Operation
2     literals powerDown, idle, receive;
3 endnewtype
4
5 signal CC2420_OP_MODE (CC2420_Operation);
6
7 output CC2420_OP_MODE (powerDown);
```

Compared to the CPU, the transceiver's potential for *energy scheduling* is rather limited. Recall that we have defined energy scheduling to be an implicit approach, relying on state information available to the SVM only. As we want to have explicit control over the transceivers inactive states, the only leeway for energy scheduling is the transition to send mode, which is implicitly signaled when a frame is transmitted. Thus, no further scheduling strategy needs to be devised and incorporated into the SVM.

5 Implementation of Energy Control Aspects

In this section, we outline how energy control aspects explicitly specified in SDL and implicitly derived during execution of SDL models are implemented on real hardware platforms. In particular, we elaborate on the implementation of energy mode signaling (Sect. 5.1) and energy scheduling (Sect. 5.2) as introduced in Sect. 4. Specific implementation aspects are explained by referring to the Imote2 hardware platform [6].

5.1 Energy Mode Signaling

Energy mode signaling is achieved by an extension of our SDL Environment Framework (SEnF) [22], which provides the interfacing of the SDL design model and the (hardware) environment on implementation level. The SEnF is a collection of drivers and routines supporting various hardware devices (e.g. IEEE 802.11 a/b/g (WLAN), IEEE 802.15.1 (Bluetooth), IEEE 802.15.4 (ZigBee), RS-232 (UART), LEDs, Netcams) and operating systems (e.g. Linux, Windows XP). Basically, special SDL signals are sent to the environment and interpreted accordingly, by setting registers of the Imote2.

CPU frequency mode changes. Signaled to the environment are consumed by the CPU driver of the SEnF. In case of the Imote2, the processor, the bus speed and the voltage for the mode change have to be selected. These values are written to the *Power Management Controller* via I^2C and to the coprocessor register `CR6`.

CPU operation mode changes. Have to be applied very carefully. According to the datasheet, certain preconditions must be fulfilled before a new mode can be selected, in order to avoid data loss or unpredicted behavior. The operation modes idle and deep idle can be entered without further considerations. In both modes, memory refresh and timer work as in active mode; only peripherals and statement execution are suspended. Wakeup is triggered by a hardware interrupt, e.g. by setting a hardware timer with a duration specified as parameter of the special SDL signal. In the remaining modes, no time-triggered wakeup is possible. For instance, in sleep or deep sleep mode, the Imote2 can only be reactivated by a hardware interaction such as pressing a button. Operation modes are processed by writing the selected mode to the coprocessor register `CR7`. In case of deep idle mode, a frequency mode change to 13 MHz is implicitly done before the operation mode change.

Transceiver mode changes. Signaled to the environment are consumed by the transceiver driver of the SEnF. Before applying mode changes, an error check regarding the current transceiver mode is applied, e.g. whether the transceiver is still sending when to be powered down. If no error is detected, a command is given to the transceiver via the SP-Interface, and the current energy mode is saved in the driver.

5.2 Energy Scheduling

Energy scheduling is achieved by an extension of the SDL virtual machine (SVM), which schedules SDL transitions and triggers their execution. Based on information available to the SVM during execution, energy modes of the CPU are chosen. In particular, the SVM has to decide when to sleep and which sleep mode to use, depending on current system activity. To leave the active mode, all schedulable transitions must have been fired, and no further signals are queued in the environment. If the first condition is satisfied, the SVM polls the environment using the interface function `xInEnv`, which has the following signature:

```
void xInEnv (SDL_Time next_timer_expiration);
```

If there is in the environment a pending signal, received (for example) from other network nodes, this signal is moved to the corresponding SDL process queue. If there is no pending signal, the parameter of `xInEnv`, which denotes the time when the next SDL timer will expire, is used to determine the next CPU operation mode, and to set a hardware timer. Let $d = (\texttt{next\_timer\_expiration} - \texttt{now})$ be the remaining duration until then. In principle, the CPU can be switched to a sleep mode if $d > 0$. However, for precise energy scheduling, we also have to consider the time it takes to reactivate the CPU. Therefore, we change the CPU operation mode from active to idle only if $d > 60\,\mu\text{s}$, and to deep idle only if $d > 2\,\text{s}$. The hardware timer is set to $t = \texttt{now} + (d - \texttt{estimated\_wakeup\_time})$. Mode changes from idle modes to active are either triggered by the expiration of the hardware timer, or by other hardware interrupts.

6 Case Study *Inverted Pendulum*

We illustrate the application of energy mode signaling and energy scheduling by our *inverted pendulum system*, a wireless networked control system (WNCS), where controllers, sensors, and actuators exchange information over a wireless digital communication network. The main challenge of this WNCS is to achieve predictable performance and stability in all possible dynamic situations. To extend the lifetime of the WNCS, energy consumption has to be reduced.

The inverted pendulum is shown in Fig. 3. The vertical rod can rotate around a fixed point on the cart. The corresponding angle from the vertical upward position is ψ. The cart displacement and velocity are x and v, respectively. A linear force u can be applied to the cart using a DC electric motor coupled to the cart through a transmission belt. The mathematical model of the system and its parameters can be found in [23].

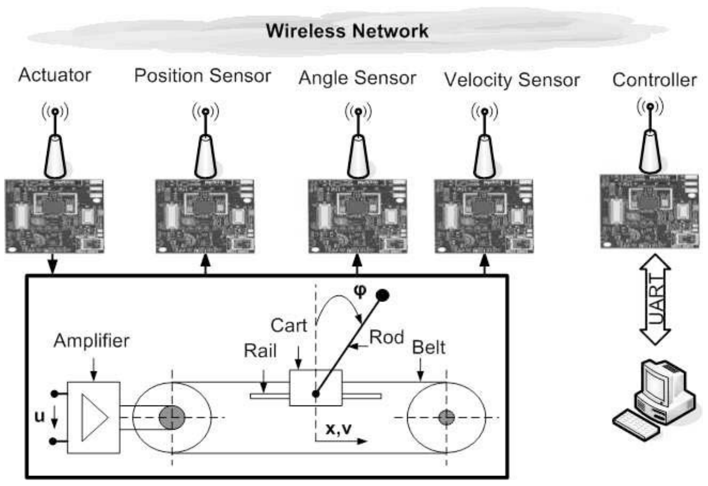


Fig. 3. Inverted pendulum with wireless network nodes

Figure 3 also shows the network topology of the inverted pendulum control system consisting of 5 Imote2 nodes communicating via IEEE 802.15.4. Attached to the pendulum are three sensor nodes measuring position and velocity of the cart, and the angle of the rod. In addition, there is a node used for the DC motor, and a node acting as controller. During operation, sensor values are transmitted periodically to the controller, which computes control values and sends them to the actuator. To keep the inverted pendulum sufficiently stable, a sampling period of 30 ms has to be assured.

For WNCSs in general, and for the inverted pendulum in particular, we have devised a specialized 4-layer communication system, with application-specific layer called *WNCS Communication Middleware* (*WNCS_CoM*) on top, using SDL as design language. The idea is that sensors and actuators register specific services (e.g., periodical value delivery), to which controllers can subscribe. This service interface supports control applications on a high level of abstraction. Service usage is illustrated by the MSC in Fig. 4, which shows a typical message scenario. Here, the control application of the sensor node registers a service called *angle*. Likewise, a service *motor* is registered on the actuator node. Afterwards, the controller subscribes to these services, providing QoS parameters specifying the sampling period. After these initializations, the sensor nodes periodically send data to the controller, which in turn gives control values to the actuator. A heartbeat mechanism completes the scenario.

All energy-relevant states and events of the angle sensor in the scenario in Fig. 4 are listed in Table 1. The entry "step" in the first column of the table refers to a relative point in time of the MSC marked by the same index. For each point in time, the previous energy scheduling and energy mode signaling events are shown. In addition, the CPU mode and transceiver mode reached at that point in time are listed. At t_1, the CPU is in active mode, running at a speed

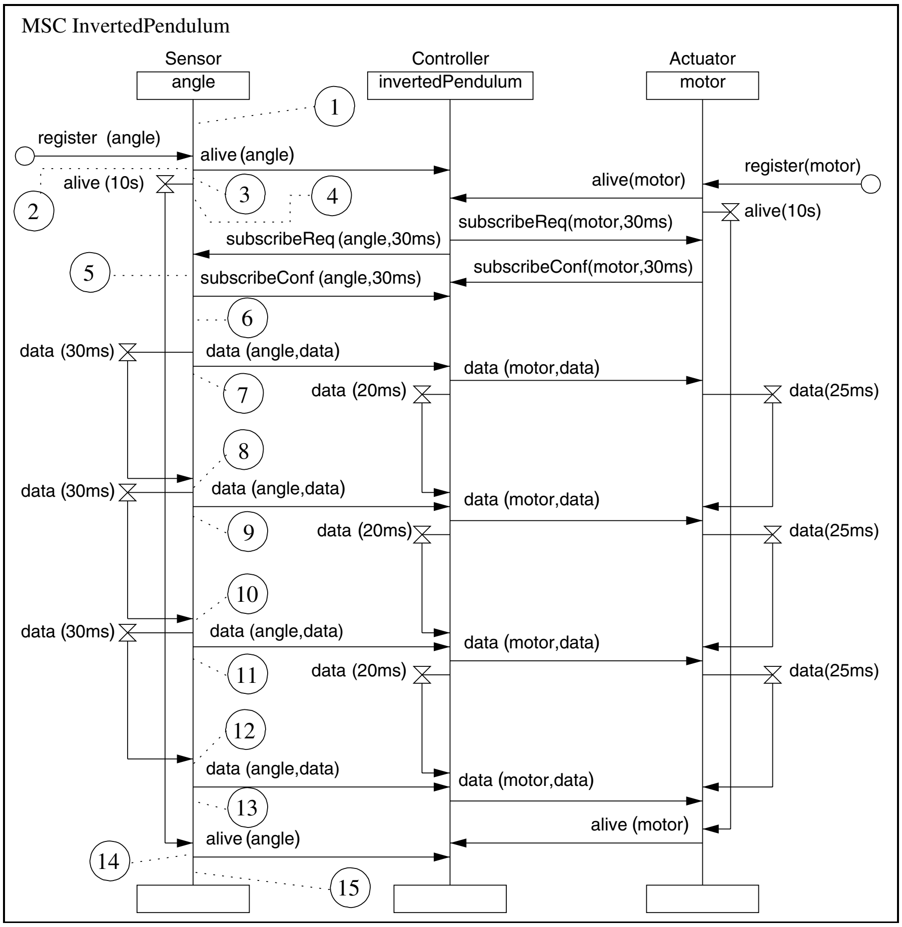


Fig. 4. MSC Inverted pendulum scenario with annotations

of 13 MHz, while the transceiver is in power down mode. By sending a signal `CC2420_OP_MODE(idle)` to the environment, the CPU switches the transceiver to idle mode. At t_3, an alive signal has been sent, so the transceiver has been implicitly switched to send mode, from which it returns to receive mode without further signaling. The CPU energy scheduler then determines that no further transitions can be executed and switches the CPU to sleep mode, while the transceiver remains in receive mode. At t_5 and t_8, receive and timer interrupts occur, respectively, and the CPU is switched to active mode again. From t_6 onward, the transceiver remains idle except when data or alive signals are to be sent. Also, the CPU is in sleep mode most of the time, reducing energy consumption of the sensor node. Similarly, the controller and actuator nodes switch between active and inactive modes.

Table 1. Energy modes of the inverted pendulum scenario for Fig. 4

step	energy scheduling	energy mode signaling	CPU mode	transceiver mode
t_1			active, 13 MHz	power down
t_2		`CC2420_OP_MODE(idle)`	active, 13 MHz	idle
t_3	→send→receive		active, 13 MHz	receive
t_4	→sleep		sleep	receive
t_5		rx interrupt	active, 13 MHz	receive
t_6	→send→receive	`CC2420_OP_MODE(idle)`	active, 13 MHz	idle
t_7	→send→receive→sleep	`CC2420_OP_MODE(idle)`	sleep	idle
t_8		timer interrupt	active, 13 MHz	idle
t_9	→send→receive→sleep	`CC2420_OP_MODE(idle)`	sleep	idle
t_{10}		timer interrupt	active, 13 MHz	idle
t_{11}	→send→receive→sleep	`CC2420_OP_MODE(idle)`	sleep	idle
t_{12}		timer interrupt	active, 13 MHz	idle
t_{13}	→send→receive→sleep	`CC2420_OP_MODE(idle)`	sleep	idle
t_{14}		timer interrupt	active, 13 MHz	idle
t_{15}	→send→receive→sleep	`CC2420_OP_MODE(idle)`	sleep	idle

The inverted pendulum case study illustrates how energy can be controlled at design time, using explicit energy mode signaling and implicit energy scheduling. In general, energy management requires a very thorough analysis to ensure that components are active when needed. In a distributed system, time synchronization among network nodes may be needed as a basis for system-wide duty cycles. These topics are not addressed here and require further study.

7 Conclusions

In this paper, we have presented two complementary approaches to specify energy aspects during the design phase, and how to incorporate them into the design language SDL. The first approach called *energy mode signaling* is based on an underlying energy model of the hardware platform, and allows transitions between energy modes of hardware components such as CPU and transceiver to be controlled explicitly, using special SDL signals. The second approach called *energy scheduling* exploits information available to the SDL Virtual Machine (SVM) at runtime to control transitions between energy modes. Since this does not require any changes of the design specification, energy scheduling is an implicit approach. We have argued that both approaches have certain limitations, which can be overcome by using them together.

Energy mode signaling and energy scheduling are straightforward to apply and to implement. We have illustrated the use of both approaches to stabilize an inverted pendulum, a wireless networked control system. Also, we have given details of how they are implemented to control energy aspects of the Imote2 hardware platform.

It would be interesting to extend SDL in order to specify further energy aspects. For instance, by defining execution tasks and assigning energy consumption to them, more advanced energy scheduling techniques could be devised. The definition of execution tasks is currently not supported by SDL, and in fact may turn out to be difficult, as this would have to establish an SDL structure that is orthogonal to the hierarchical system architecture consisting of agents and their refinements. Also, it has to be studied how, for each task, energy consumption can be determined. In an SDL specification, the execution time - and therefore the energy consumption - of the SDL Virtual Machine remains implicit. This would also be the case if execution is structured into individual tasks.

References

1. Akyildiz, I.F., Sankarasubramaniam, Y., Cayirci, E.: Wireless Sensor Networks: A Survey. Computer Networks 38(4), 393–422 (2002)
2. Tonguz, O.K., Ferrari, G.: Ad Hoc Wireless Networks: A Communication-theoretic Perspective. John Wiley & Sons, Chichester (2006)
3. Walsh, G.C., Ye, H., Bushnell, L.: Stability Analysis of Networked Control Systems. IEEE Transactions on Control Systems Technology 10(3), 438–445 (2002)
4. International Telecommunications Union: Recommendation Z.100(11/07), Specification and Description Language (SDL), http://www.itu.int/rec/T-REC-Z.100/en
5. Object Management Group (OMG): Unified Modeling Language (UML), http://www.uml.org
6. Crossbow: Datasheet, http://xbow.com/Products/Product_pdf_files/Wireless_pdf/Imote2_Datasheet.pdf
7. Rao, R., Vrudhula, S., Rakhmatov, D.N.: Battery modeling for energy aware system design. Computer 36(12), 77–87 (2003)
8. Benini, L., Macii, A., Poncino, M.: Energy-aware design of embedded memories: A survey of technologies, architectures, and optimization techniques. ACM Transaction on Embedded Computing Systems 2(1), 5–32 (2003)
9. Aydin, H., Melhem, R., Mosse, D., Majia-Alvarez, P.: Power-aware scheduling for periodic real-time tasks. IEEE Transactions on Computers 53(5), 584–600 (2004)
10. Herczeg, Z., Schmidt, D., Kiss, A., Wehn, N., Gyimothy, T.: Energy Simulation of Embedded XScale Systems with XEEMU. Journal of Embedded Computing, Special Issue 3(3) (2009)
11. Ye, W., Heidemann, J.S., Estrin, D.: An energy-efficient MAC protocol for wireless sensor networks. In: Proceedings IEEE INFOCOM 2002, pp. 1567–1576 (2002)
12. Du, S., Saha, A.K., Johnson, D.B.: RMAC: A Routing-Enhanced Duty-Cycle MAC Protocol for Wireless Sensor Networks. In: Proceedings IEEE INFOCOM 2007, pp. 1478–1486 (2007)
13. Akkaya, K., Younis, M.: A Survey on Routing Protocols for Wireless Sensor Networks. Ad Hoc Networks (Elsevier) 3(3), 325–349 (2005)
14. Mohsen, A., Hofmann, R.: Efficient Voltage Scheduling and Energy-aware Co-synthesis for Real-time Embedded Systems. In: Srikanthan, T., Xue, J., Chang, C.-H. (eds.) ACSAC 2005. LNCS, vol. 3740, pp. 3–14. Springer, Heidelberg (2005)
15. Telelogic AB: Telelogic Tau Generation 1 - SDL Tau Suite, http://www.telelogic.com/products/tau/

16. Fliege, I., Grammes, R., Weber, C.: ConTraST - A Configurable SDL Transpiler and Runtime Environment. In: Gotzhein, R., Reed, R. (eds.) SAM 2006. LNCS, vol. 4320, pp. 216–228. Springer, Heidelberg (2006)
17. Intel: PXA27x Documents, `http://int.xscale-freak.com/XSDoc/PXA27X/PXA27X_index.htm`
18. Yale School of Engineering & Applied Science: Power Modes and Energy Consumption for the iMote2 Sensor Node. Technical report, Enalab, `http://enaweb.eng.yale.edu/drupal/system/files/imote2_power.pdf`
19. Chipcon / Texas Instruments: CC2420 datasheet, `http://focus.ti.com/lit/ds/symlink/cc2420.pdf`
20. Glässer, U., Gotzhein, R., Prinz, A.: The Formal Semantics of SDL-2000 - Status and Perspectives. Computer Networks 42(3), 343–358 (2003)
21. ITU-T Recommendation Z.100 Annex F (11/00), SDL Formal Definition, `http://www.itu.int/rec/T-REC-Z.100/en`
22. Kuhn, T., Gotzhein, R., Webel, C.: Model-Driven Development with SDL - Process, Tools, and Experiences. In: Nierstrasz, O., Whittle, J., Harel, D., Reggio, G. (eds.) MoDELS 2006. LNCS, vol. 4199, pp. 83–97. Springer, Heidelberg (2006)
23. Gabel, O., Litz, L.: NCS Testbed for Ambient Intelligence. In: Proceedings of the IEEE SMC Conference, pp. 115–120 (2005)

Model-Driven Development of Time-Critical Protocols with SDL-MDD

Philipp Becker, Dennis Christmann, and Reinhard Gotzhein

Networked Systems Group, University of Kaiserslautern, Germany
{pbecker,christma,gotzhein}@cs.uni-kl.de

Abstract. Contention-based medium access in wireless networks suffers from the problem of frame collisions. In previous work, we have introduced new transfer protocols for the network-wide transmission of bit sequences that overcome the problem of destructive collisions. In this paper, we present the model-driven development of these protocols with SDL-MDD and its SDL tool chain. On the one hand, we show how to formally specify low-level functionality and time-critical behavior – network-wide deterministic arbitration – using the available constructs of a high-level design language. On the other hand, we show the embedding of this high-level design into our SDL execution environment that is extended to support time-critical requirements of the introduced transfer protocols.

Keywords: SDL, SDL-MDD, black burst, cooperative/arbitrating transfer protocol, MacZ.

1 Introduction

Technological advances over the past decade have led to the wide dissemination of mobile devices, communicating over wireless links. Mobile phones, laptop computers, and motes[1] have opened up new markets and have changed our everyday life. A main driver for this success is the progress in radio communication, ranging from efficient coding schemes and more powerful, yet energy-efficient transceivers to sophisticated protocols supporting wireless media. A common property of these protocols is that they have to arbitrate a shared medium. This is usually done either by adopting a point coordinator, i.e. a node that schedules transmissions of other nodes, or by contention-based access.

Contention-based medium access suffers from the problem of frame collisions, where the information carried by a data frame may be destroyed due to interferences. Several measures such as listening before transmitting (CSMA), non-deterministic medium access delays (random backoff), and 2-hop short-term reservations (RTS/CTS) reduce the problem, but cannot solve it, always leaving a non-deterministic element.

[1] Common synonym for wireless sensor nodes.

In previous work [1], we have introduced new protocols for the network-wide transfer of bit sequences in wireless networks, which overcome the problem of destructive collisions. *Arbitrating transfer* supports network-wide arbitration such that exactly one node wins after a fixed time. *Cooperative transfer* can propagate a common bit sequence such as a time value or a node address across the entire network, with deterministic delay. Both protocols are based on encodings with so-called *black bursts*, which are special transmissions insensitive to interference and therefore protected against collisions.

In this paper, we study the model-driven development of the protocols for the network-wide transmission of bit sequences with black bursts, using SDL [2] as design language. We apply SDL-MDD [3,4], our model-driven development process, and its SDL tool chain, which in particular supports the automatic generation of code for embedded systems directly from the SDL design model. The specific challenge on the one hand is how to formally specify low-level functionality and time-critical behavior, i.e. network-wide deterministic arbitration based on black bursts, using the available constructs of a high-level design language. On the other hand, we show how our extended SDL execution environment supports time-critical requirements of the arbitrating transfer protocols. By means of our SDL tool chain, automatic implementations can be obtained that are embedded into the extended execution environment.

The paper is structured as follows. In Sect. 2, we survey related work. Section 3 explains the ideas of arbitrating transfer of bit sequences in wireless networks. Section 4 is the core of the paper, presenting the model-driven development of arbitrating transfer with SDL-MDD. Section 5 supplements Sect. 4 by introducing further measures at implementation level that help to handle time-critical behavior. In Sect. 6, we draw conclusions and outline future work.

2 Related Work

Related work can be grouped into two categories: the design of low-level functionality with SDL, and the implementation of SDL design specifications. We do not consider work on arbitration protocols or on channel encoding as related work, as this is not the focus of the paper.

The design of low-level functionality with SDL has received little attention so far. Mohsen et al. [5] present an integrated methodology and tool support for voltage scheduling and energy-aware co-synthesis of real-time systems. It is mentioned that SDL has been used; however, this is not substantiated in the paper.

Regarding the implementation of SDL design specifications, the situation is entirely different. Several SDL compilers developed in industry (e.g. Cadvanced, Cmicro [6], Real Time Developer Studio [7]) and academia (e.g. ConTraST [8]) contribute to the automatic code generation. A major part of the textbook [9] is devoted to the implementation of SDL, elaborating on different strategies regarding, e.g., timer management, input queue management, and interfacing with the environment. The latter is of particular importance when it comes to a mapping of encoding events to the underlying operating system and hardware platform. Since this is a very broad field of possible implementation choices, we will not go into further detail at this point.

In [10], the implementation of SDL specifications is also addressed. Beside the general problems due to gaps between assumptions in SDL and real world conditions (e.g., consideration of processing time, availability/capacity of resources), Bræk et al. discuss differences between SDL and modern programming languages such as built-in support for concurrency, time, and communication. To fulfill real-time constraints, prioritization of processes and messages must be introduced that are currently not supported by SDL constructs and must be considered at implementation level.

Optimization of SDL systems at implementation level is also discussed in [11], especially the usage of priority assignments to processes, which are not covered in SDL, but in many tool chains. Also by softening the SDL *copy by value* semantics in signal parameters, a performance gain can be achieved, because in many cases, copying of signal parameters into process variables represents avoidable overhead when SDL signals are forwarded only.

3 Arbitrating Transfer of Bit Sequences in Wireless Networks

In wireless networks, destructive frame collisions may occur if several transmissions overlap in time and place. To reduce the probability of frame collisions during contention access periods, MAC protocols usually adopt several measures such as CSMA, random backoff, and RTS/CTS schemes; however, this does not solve the problem of destructive frame collisions.

To solve the problem, we have proposed new protocols for the network-wide transfer of bit sequences, located at MAC level [1]. These protocols are based on encodings with so-called *black bursts*, which are special transmissions insensitive to interference and therefore protected against collisions. More precisely, a black burst is a period of transmission energy of defined length on the medium. Transmission of black bursts occurs at determined points in time, without prior medium arbitration. If two or more nodes transmit a black burst at (almost) the same time, a receiving node can still detect the period of energy on the medium including starting and ending time. Since this is the only information carried by a black burst, the collision is non-destructive.

In [12], we have implemented black bursts by MAC frames without payload, transmitted without prior medium arbitration. Start and end of reception are detected by using the clear channel assessment (CCA) mechanism of the transceiver.

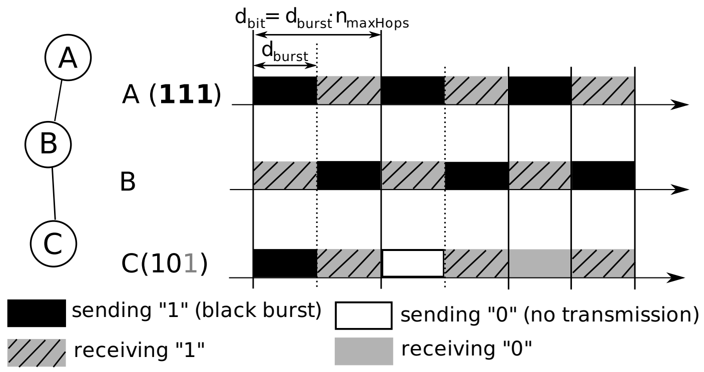
Bits can be encoded with black bursts of different length. Below, we use black bursts of length $d_b > 0$ to encode a binary 1, and of length $d_b = 0$ (i.e. no transmission) to encode a binary 0. This stipulation yields two important properties. First, a binary 1 is dominant, i.e. if two nodes transmit different bit values at (almost) the same time, the binary 1 is received by nodes in range.

Second, a node "transmitting" a binary 0 is actually not in transmission mode, but can listen on the medium. Thus, it can detect whether another node in range is transmitting a binary 1.

To avoid destructive collisions in the presence of multiple simultaneous transmissions, black burst encodings in combination with specific transfer protocols can be used. In [12], we have proposed a protocol for multi-hop tick synchronization, where all network nodes agree on joint reference points in time called *ticks*. Tick synchronization is a sufficient basis for system level applications such as network-wide medium slotting and duty cycling. In [1], we have introduced two further protocols for network-wide transfer of bit sequences, referred to as cooperative transfer and arbitrating transfer. With *cooperative transfer*, it is possible to propagate a common value, for instance, a time value or a node address, across the network in deterministic time. We have used this protocol to perform master-based *time* synchronization in tick-synchronized networks. With *arbitrating transfer*, it is possible to achieve network-wide arbitration during contention periods, such that exactly one node wins after a fixed time.

We now describe arbitrating transfer of bit sequences, which we explain by means of the example in Fig. 1. The example shows a topology consisting of three nodes, where nodes A and C start transmitting bit sequences 111 and 101, respectively. We also refer to these bit sequences as *arbitration sequences*, in line with the purpose of the transfer. Arbitration sequences could, for instance, be unique message identifiers, chosen according to the priority of the message, or node identifiers. It is crucial that transmission of arbitration sequences starts at (almost) the same point in time. This can be achieved by an underlying tick synchronization establishing network-wide reference points in time (see [12]), and by agreeing on a joint schedule for arbitration phases relative to these reference points.

During an arbitration phase, there are two types of nodes. We will refer to a node that is in the process of sending its own arbitration sequence as TRANSMITTER. A node not sending its own arbitration sequence is called REPEATER. In Fig. 1, nodes A and C are TRANSMITTERs at the beginning of the arbitration



$d_{bit} = d_{burst} \cdot n_{maxHops}$
d_{burst}
A
B
C
A (**111**)
B
C(101)
sending "1" (black burst)
sending "0" (no transmission)
receiving "1"
receiving "0"

Fig. 1. Example: Network-wide arbitrating transfer of a bit sequence

phase, node B is a Repeater. Note that a Transmitter becomes a Repeater as soon as it loses the arbitration, but a Repeater never turns into a Transmitter in the same arbitration phase.

The diagram in Fig. 1 is structured into *black burst intervals* of length d_{burst} and *bit intervals* of length d_{bit}. In the first black burst interval, nodes A and C, initially being Transmitters, send the first bit of their arbitration sequences, i.e. a binary 1 encoded as a black burst. Node B, being a Repeater, listens on the medium during this interval and detects the black burst. Since the network has a diameter of 2 hops, a Repeater receiving a black burst in the first black burst interval has to forward this to the remaining nodes in the subsequent black burst interval. So, in the second black burst interval, node B has to repeat the black burst (which is not necessary in the given topology, however, this topology may not be known to the nodes). This finishes the network-wide propagation of the first arbitration bit, which takes one bit interval. In general, $n_{maxHops}$ bit intervals are needed for each arbitration bit, where $n_{maxHops}$ denotes the maximum network diameter in hops. It follows that $d_{bit} = n_{maxHops} \cdot d_{burst}$.

Transmissions of the second bit of the arbitration sequences start in the second bit interval, which is the third black burst interval in Fig. 1. As before, nodes A and C are Transmitters, encoding a binary 1 and a binary 0, respectively. Node B listens during this interval and receives a binary 1, which is dominant. Again, node B has the task of forwarding the received bit value in the following black burst interval. Thus, the binary 1 is propagated to node C in the fourth black burst interval. Node C now recognizes that it has lost the arbitration, as another node is transmitting a bit sequence that dominates its own arbitration sequence. Therefore, it gives up its role as Transmitter and becomes a Repeater for the rest of the arbitration phase.

Arbitration is completed by transmission of the third bit. At this point, there is only one node remaining as Transmitter. Note that the duration of the arbitration phase is precisely determined as the product of the length of the arbitration sequence n_{bit}, the maximum network diameter in hops $n_{maxHops}$, and the duration of a black burst interval d_{burst}. If nodes use different arbitration sequences, exactly one node will be selected during the arbitration process. This mechanism can be used for many purposes, such as deterministic master election or network-wide medium arbitration with subsequent collision-free transmission of regular data frames in a reserved time interval.

4 Model-Driven Development of Arbitrating Transfer with SDL-MDD

In this section, we present the model-driven development of network-wide arbitrating transfer of bit sequences in wireless networks (see Sect. 3) using SDL-MDD. We briefly survey SDL-MDD, identify general guidelines for time-critical specifications, give an overview of the system architecture, and then focus on behavioral aspects. In particular, we show how low-level behavior can be specified from a conceptual point of view.

4.1 Model-Driven Development with SDL-MDD

SDL-MDD [4] is a holistic, model-driven development approach where the formal system model guides and directs all development activities, with ITU-T's Specification and Description Language (SDL) [2] as design language. It addresses all phases of system development and is supported by design methodologies and a semantically integrated and complete tool chain. Thus, SDL-MDD supports rapid development, increases productivity through reusability, and improves quality.

SDL-MDD distinguishes two phases, the specification phase and the implementation phase. The steps of the *specification phase*, which is related to OMG's MDA [13], are shown in Fig. 2. It starts with the definition of the computation-independent model (CIM), where message sequence charts (MSC [14]) are used to collect requirements and to define functionalities. In the second step, the platform-independent model (PIM) is constructed using SDL as a design language. The PIM is functionally complete and can therefore already be analyzed and validated. However, it abstracts from platform-dependent details such as the association with a concrete environment. In the third step, the PIM is extended by adding platform-specific details, yielding the platform-specific model (PSM). In particular, the PSM is obtained by selecting a hardware platform, adding corresponding abstract hardware interfaces to the PIM, and by configuring hardware parameters.

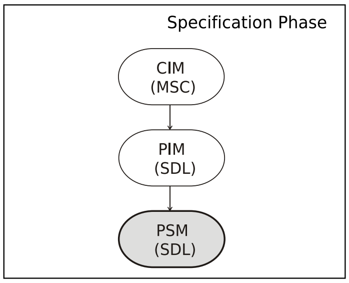


Fig. 2. SDL-MDD [4]

Productivity and quality of model-driven development with SDL-MDD are increased by using SDL design patterns and SDL micro protocols [15,4] throughout the specification phase. An *SDL design pattern* is a fine-grained, reusable generic solution for a recurring design problem. An *SDL micro protocol* is a communication protocol with a single, distributed functionality and the required protocol collaboration. When designing the protocols for cooperative and arbitrating transfer of bit sequences (see Sect. 3), we have defined and composed several micro protocols, yielding a high-quality system structure that is highly maintainable (see Sect. 4.3).

In the *implementation phase* of SDL-MDD, executables are automatically generated from the PSM in two steps. In the first step, intermediate code in languages such as C or C++ is generated, using available code generators (e.g. Tau Cadvanced [6], ConTraST [8]). This code is then compiled with a platform-specific C/C++-compiler, augmented by hardware drivers of our SDL Environment Framework (SEnF [4]), and executed under the control of the SDL Virtual Machine (SVM). In addition, it is possible to compile an executable for performance simulations, too.

4.2 Guidelines for Time-Critical Specifications

When specifying low-level and time-critical systems with SDL, we have to deal with delays that may occur until a fireable transition is selected and executed. Basically, there are two reasons why a transition may be delayed, although there is a corresponding stimulus in the input queue. First, the stimulus may be enqueued behind other stimuli that are processed before. Second, the execution time of a running transition is not determined. Since transition execution is atomic, other fireable transitions are delayed until the active transition is completed (*sequentialization delay*). Thus, the delay of a fireable transition is in general unknown and might be unbounded, e.g., if loops are used.

To deal with this problem, we have defined and adopted the following specification guidelines:

1. To accelerate the consumption of time-critical signals, we constrain the system architecture by stipulating that designers use short communication paths for time-critical signals. This means that SDL components exchanging time-critical signals should be placed topologically close to each other, avoiding intermediate processes if possible. In particular, SDL components exchanging time-critical signals with the environment should be directly attached to the system border.
2. To further accelerate the consumption of time-critical signals, we require them to be declared as *priority input*. According to the semantics of SDL, signals declared as priority input are processed before regular signals and continuous signals. A drawback of SDL-2000 is that there is only one priority, which inhibits the distinction of different degrees of time-criticality. Priority inputs should be used moderately in order to be effective.
3. To cope with the problem of sequentialization delay, complex transitions with long execution times are to be avoided. If necessary, complex computations should be split into several transitions, reducing the sequentialization delay of time-critical transitions.
4. Finally, to achieve fast firing times of time-critical transitions, the SDL system must not get overloaded during execution. This requires the choice of a suitable hardware platform and the analysis of the overall system behavior, based on an offline performance analysis. In [10], Bræk et al. suggest the calculation of mean peak load and advise optimization of software and/or speed-up of hardware if the mean peak load exceeds 0.3 to allow statistical variation.

Yet, these measures cannot guarantee that all real-time requirements can be met. To further improve execution time and accuracy, additional measures can be applied by introducing optimizations at the implementation level. Section 5 addresses these measures in detail by presenting enhancements of the execution environment SEnF and SVM.

4.3 MAC Layer Architecture

This subsection gives an overview of the architecture of the MAC layer protocol, and in particular of our protocols for the network-wide, collision-protected

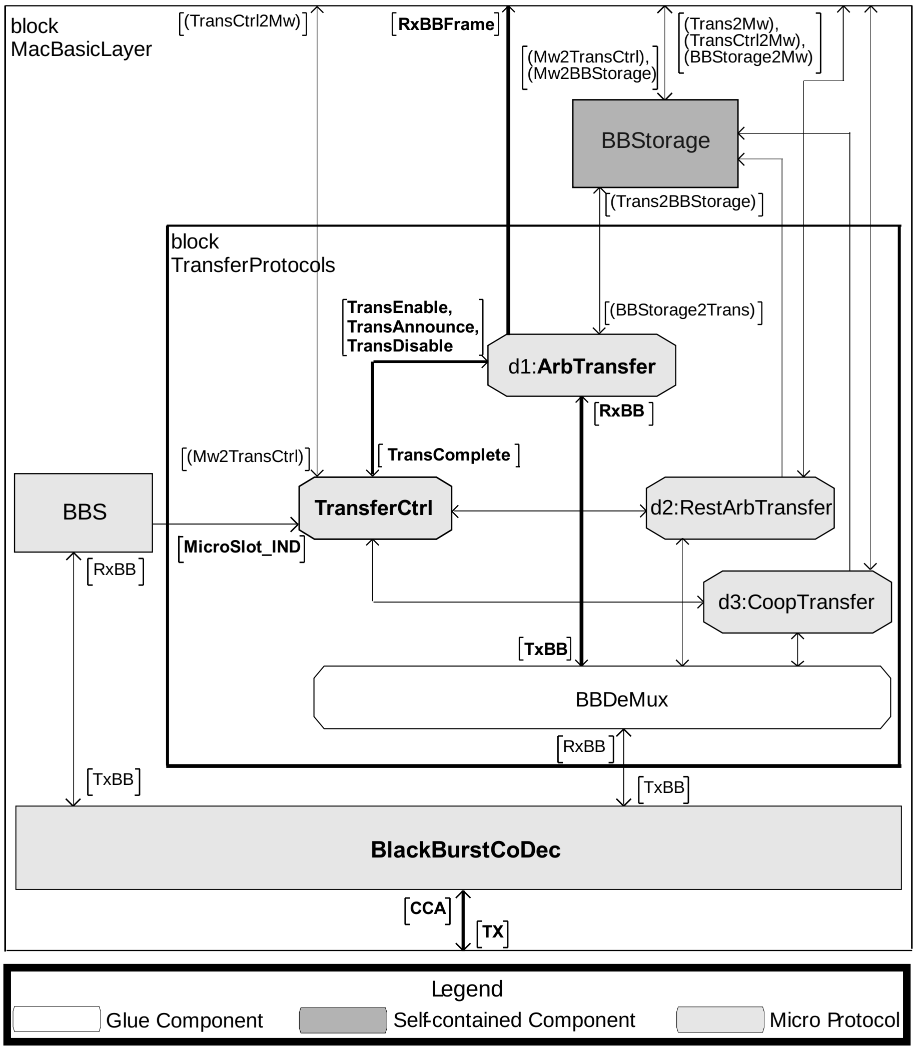


Fig. 3. Architecture of the MAC layer, including transfer protocols (Non-relevant signals are omitted)

transfer of bit sequences in wireless networks. The architecture is shown as an SDL overview diagram in Fig. 3. The complete functionality is obtained by defining and composing several SDL micro protocols, which is also shown in the figure. We will elaborate on the behavior of several of these micro protocols in the following subsections.

The architecture shows an SDL block `MacBasicLayer` and its decomposition into four blocks. `BBS` is the design of Black Burst Synchronization, our protocol for tick synchronization (see [12]). Bit sequences to be transmitted by cooperative or arbitrating transfer are managed by BBStorage. BlackBurstCoDec is responsible for encoding and decoding single black bursts, which is a low-level functionality needed by several other micro protocols. Note that being time-critical, BlackBurstCoDec is placed right on top of the hardware environment, in accordance with our specification guidelines above. The block TransferProtocols is the result of composing the transfer protocols ArbTransfer, RestArbTransfer, and CoopTransfer, the control component TransferCtrl, and BBDeMux. TransferCtrl coordinates the active periods of the transfer protocols, and for synchronizing the beginning of these active periods to the network-wide ticks obtained through BBS. BBDeMux collects and distributes outgoing and incoming black bursts, respectively. This leads to an encapsulation of TransferProtocols, which allows extensions with additional black burst-based transfer protocols without changing the interface of TransferProtocols.

4.4 BlackBurstCoDec – Creation and Detection of Black Bursts

The micro protocol BlackBurstCoDec builds the bridge between the conceptual view on black bursts and their concrete implementation. It is positioned close to the SDL environment, i.e. signals coming from and going to entities of this protocol are renamed by an additional driver component, but not processed any further. This is in accordance with our guidelines for time-critical specifications and helps to keep delay due to the signal path low. In addition, the driver component prioritizes the SDL signals CCA and TX exchanged between BlackBurstCoDec and environment.

BlackBurstCoDec itself consists of two components (not shown in Fig. 3): BlackBurstEncode is responsible for transmitting black bursts over the medium, BlackBurstDecode detects incoming black bursts. At the design level, the specification of black burst transmissions is straightforward. When triggered, BlackBurstEncode outputs an SDL signal TX to the environment, which triggers the physical transmission of a regular MAC frame of predefined length without prior CCA. Here, we may note that this MAC frame is only constructed once during initialization, and then used for every subsequent black burst transmission. This reduces runtime complexity and thus improves performance.

BlackBurstDecode is responsible for detecting single incoming dominant black bursts, independent of how they are used by a specific transfer protocol, and to forward them to BBDemux. Detection of black bursts is done by monitoring the energy on the medium, using the clear channel assessment (CCA) mechanism. Each time the mediums status changes from idle to busy or vice versa, the SDL environment sends an SDL signal CCA, containing the new medium status and a timestamp, which is consumed by BlackBurstDecode. Based on these signals, the duration of the medium's busy periods can be calculated. If the duration lies within well defined bounds, the reception of a black burst (dominant 1) is assumed and forwarded to BBDemux. The duration bounds as well as the transmission length of a black burst are chosen such that the following constraints are satisfied:

1. Slightly overlapping black bursts from different stations transmitting concurrently must still be detected.
2. Regular data frames must not be mistaken for black bursts, i.e. they must differ significantly in length.
3. Inaccuracies of the (hardware-related) CCA mechanism must not prevent black bursts from being detected. This includes also the attenuation of hysteresis performed by the transceiver.

As the accuracy of black burst timing is crucial for tick synchronization with `BBS` and for the transfer protocols using black burst encodings, we have to be very careful when specifying this mechanism. In particular, we have to remove factors that may decrease the accuracy of detecting black bursts as far as possible. Among these factors are, for instance, delayed environment polling, SDL signal delay, and late timestamping.

In a previous SDL design of `BlackBurstDecode`, we used SDL time to create timestamps. In that design, the current value of `now` was associated with incoming `CCA` signals that are created by the environment when a medium status change is detected. It is quite obvious that this high-level solution may be too inaccurate, as there exist unknown delays between the detection of a status change of the medium on hardware level and the consumption of the corresponding SDL signal. Following our guidelines for time-critical specifications only alleviates rather than solves this problem.

Using early timestamping, i.e. saving the time value from when the medium status change was detected by the hardware, would represent a better solution. Therefore, we decided to rely on hardware-level implementations of some time-critical aspects, which we have added to our SDL execution environment (see Sect. 5).

4.5 TransferCtrl – Scheduling of Transfer Protocols

`TransferCtrl` is the control component responsible for the timely activation and deactivation of cooperative and arbitrating transfer protocols. It manages requests for the cooperative and arbitrating transfer of bit sequences, and schedules so-called *transfer opportunities*, i.e. time intervals where black burst encodings may be used. A transfer request consists of the PId of the calling process, a unique job ID, the type of transfer protocol, and the length of the bit sequence that should be sent or received. Actual bit sequences are not part of the request managed by `TransferCtrl`, but retrieved by the transfer protocol from `BBStorage`.

The activation of a particular transfer protocol consists of two steps. First, the upcoming activation is announced to the transfer protocol well in advance, via the SDL signal `TransAnnounce` communicating a particular transfer request. This early announcement is not time-critical and decouples the preparation of the transfer from the actual, time-critical transfer. The next pending request may already be announced to the corresponding transfer protocol (`ArbTransfer`, `RestArbTransfer`, or `CoopTransfer`). Then, `TransferCtrl` waits for the next

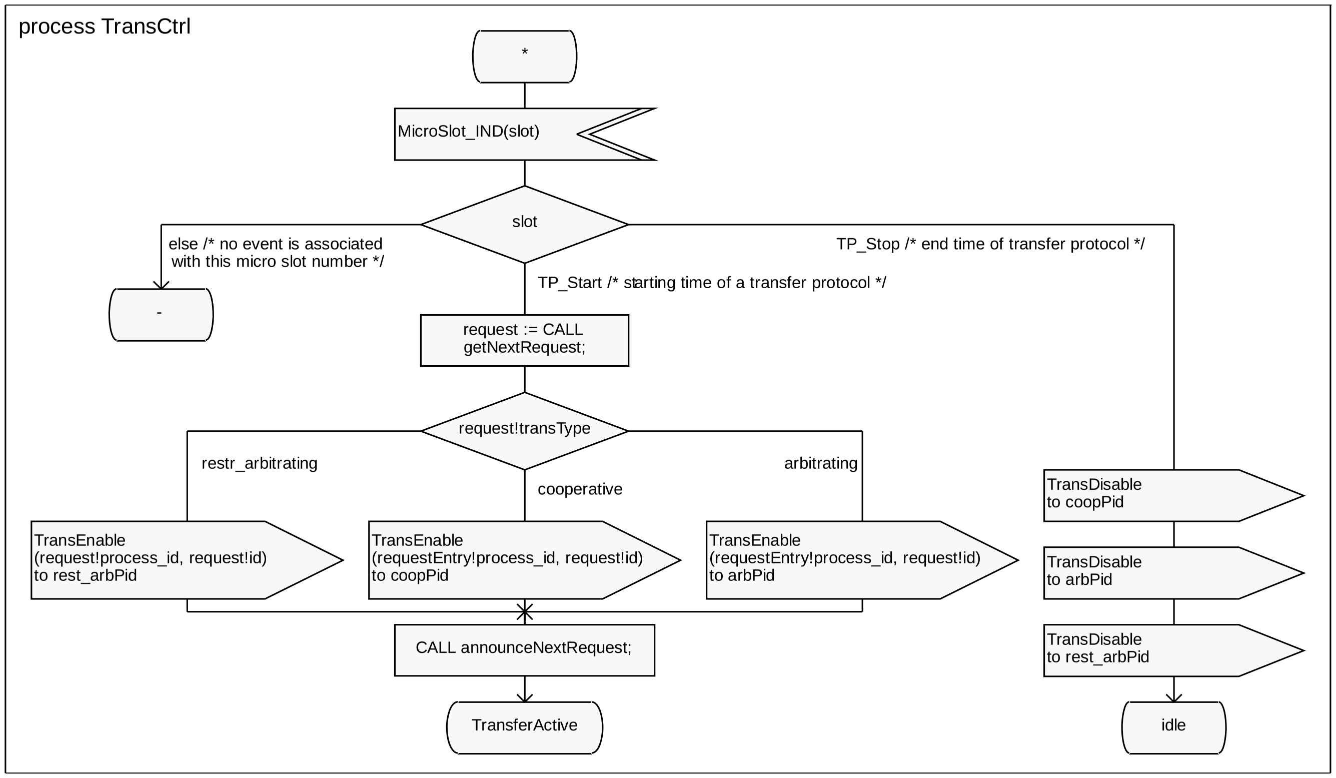

process TransCtrl
*
MicroSlot_IND(slot)
slot
else /* no event is associated with this micro slot number */
-
TP_Start /* starting time of a transfer protocol */
TP_Stop /* end time of transfer protocol */
request := CALL getNextRequest;
request!transType
restr_arbitrating
cooperative
arbitrating
TransEnable (request!process_id, request!id) to rest_arbPid
TransEnable (requestEntry!process_id, request!id) to coopPid
TransEnable (requestEntry!process_id, request!id) to arbPid
CALL announceNextRequest;
TransferActive
TransDisable to coopPid
TransDisable to arbPid
TransDisable to rest_arbPid
idle

Fig. 4. `TransferCtrl` (excerpt) – Scheduling of transfer protocols transfer opportunity. Transfer opportunities are associated with micro slots of the underlying medium slotting, which are determined by BBS, the tick synchronization protocol. Between two ticks, micro slots are consecutively numbered, starting with 1. BBS signals the start of each micro slot to TransferCtrl by SDL signals MicroSlot_IND. In Fig. 4, the transition consuming these signals is shown. Since signal consumption is time-critical, a priority input is specified, in accordance with our guidelines.

If the signaled micro slot number corresponds to the beginning of the next transfer opportunity, the next scheduled request is determined, and the corresponding transfer protocol is enabled by sending an SDL signal TransEnable (see Fig. 4). Retrieval of the bit sequence and its transfer is then entirely under the control of the transfer protocol (see Sect. 4.6). When the request has been processed, i.e. when a bit sequence has been sent or received, the transfer protocol will return an SDL signal TransComplete. If at this point, another request is pending, the corresponding transfer protocol is enabled. This continues until either all requests have been handled or the transfer opportunity terminates that is again associated with a specific micro slot number. At this point, all transfer protocols are disabled (see Fig. 4).

4.6 ArbTransfer – Arbitrating Transfer Protocol

The micro protocol ArbTransfer performs the arbitrating transfer of bit sequences, using black burst encoding. When activated by TransferCtrl, it divides the time into bit and burst intervals, and starts sending black bursts as described in Sect. 3. When activated, ArbTransfer either assumes the role of TRANSMITTER, sending its own arbitration sequence, or REPEATER, forwarding received black bursts in the next black burst interval. We will now elaborate on the SDL design of ArbTransfer.

For the timely behavior of ArbTransfer, synchronicity of the starting points of burst intervals is crucial. Therefore, we have removed all avoidable delays in our specification that could defer the starting points of both the first burst interval and all subsequent ones. In particular, we distinguish between a setup phase and a transfer phase. The *setup phase* is triggered by the announcement of an upcoming activation via the SDL signal TransAnnounce. This decouples the preparation of the transfer from the actual, time-critical transfer. During this phase, ArbTransfer retrieves the bit sequence of the next request from BBStorage, thereby becoming TRANSMITTER at the beginning of the next transmission opportunity. If no bit sequence is retrieved, ArbTransfer becomes REPEATER. In the *transfer phase*, triggered by the reception of the SDL signal TransEnable from TransferCtrl, the bit sequence is processed.

To achieve high accuracy during the transfer phase, we save the start time t_{phase} of the current phase as local reference point in time, and use it to determine the start times of subsequent burst and bit intervals (see Sect. 3). More specifically, we set an SDL timer that expires at the beginning of the next burst interval, by computing $t_{phase} + (i-1) \cdot d_{burst}$ as expiration time, where i denotes

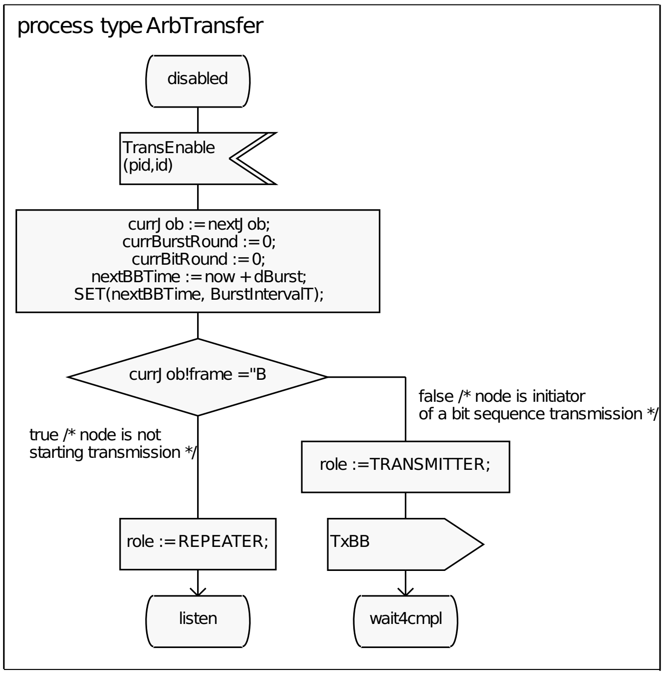


Fig. 5. `ArbTransfer` (excerpt 1)

the number of the current black burst interval. Thus, delays that may occur due to consumption of timer signals do not sum up, as they would in case of using $\texttt{now} + d_{burst}$ as the next timer setting.

To clarify the functionality of `ArbTransfer` and the application of our design rules, we present some transitions of our SDL specification.

Figure 5 shows the start of the arbitrating transfer phase, which is indicated by `TransferCtrl` by the time-critical signal `TransEnable`, received via a priority input. In this transition, `ArbTransfer` initializes context variables, saves the starting time of the current phase, sets the burst interval timer `BurstIntervalT` to `now` $+dBurst$ (the old value being irrelevant at this point), and decides whether it takes the role of Transmitter or Repeater depending on the availability of an arbitration sequence. If `ArbTransfer` takes the role of Transmitter, it sends a dominant start of frame bit (SOF, not part of the arbitration sequence) and changes the state to `wait4cmpl`. In `wait4cmpl`, `ArbTransfer` ignores incoming black bursts and waits until the next bit interval begins. Otherwise, if the Repeater role is taken, `ArbTransfer` changes to state `listen`, where it waits for incoming black bursts to be forwarded.

In Fig. 6, the reaction on an incoming black burst in state `listen` is shown, which is time-critical, too. Here, the `ArbTransfer` component extends the received bit sequence that represents a binary 1, and becomes Repeater. Note

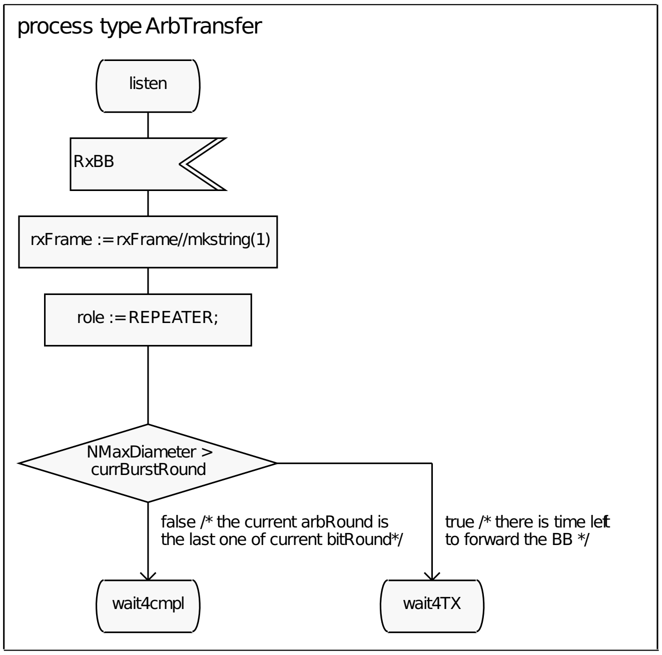

process type ArbTransfer
listen
RxBB
rxFrame := rxFrame//mkstring(1)
role := REPEATER;
NMaxDiameter > currBurstRound
false /* the current arbRound is the last one of current bitRound*/
true /* there is time left to forward the BB */
wait4cmpl
wait4TX

Fig. 6. `ArbTransfer` (excerpt 2)

that the component could already be a Repeater, because `listen` is entered both if `ArbTransfer` is a Transmitter and sending a binary 0, or if it is a Repeater. In all cases, `ArbTransfer` decides whether there is a black burst interval left to forward the black burst (waiting for this interval in state `wait4TX`), or if the black burst was received in the last black burst interval of the current bit interval (going into state `wait4cmpl`).

Finally, Fig. 7 depicts the behavior in state `wait4cmpl` when the burst interval timer is consumed. Here, `ArbTransfer` checks if the elapsed burst interval was the last one of the bit interval. If this is not the case, the component awaits another burst interval by staying in state `wait4cmpl` and setting the burst interval timer with *nextBBTime* set to $nextBBTime + dBurst$ (rather than `now` $+dBurst$) to avoid time drift and achieve accuracy. On the other hand, if the bit interval is finished, `ArbTransfer` checks whether the elapsed bit interval represents the last interval of the arbitration sequence. If this is the case, `TransferCtrl` informs about the end of transfer and forwards the received bit sequence to the requesting SDL process. Otherwise, it processes the next bit of the bit sequence that depends on the current role and sets the burst interval timer again. We should mention that the bit sequence seen on the medium is always forwarded to the requesting SDL process at the end of the transfer protocol, even if the station has transmitted this particular bit sequence itself. This is done because the requesting process can then compare its bit sequence with the received bit sequence and decide if it has won the arbitration.

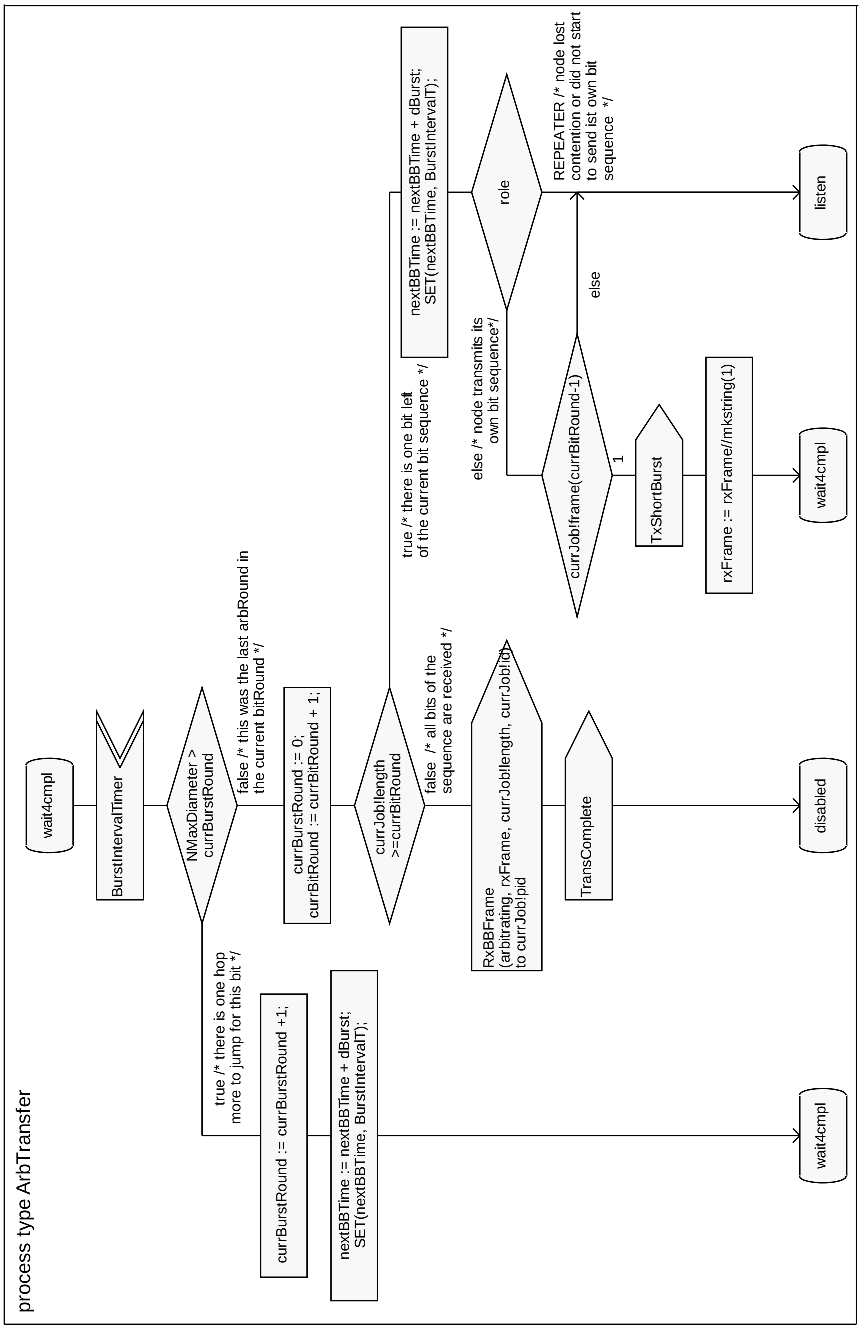

process type ArbTransfer
wait4cmpl
BurstIntervalTimer
NMaxDiameter > currBurstRound
true /* there is one hop more to jump for this bit */
false /* this was the last arbRound in the current bitRound */
currBurstRound := currBurstRound +1;
nextBBTime := nextBBTime + dBurst; SET(nextBBTime, BurstIntervalT);
wait4cmpl
currBurstRound := 0; currBitRound := currBitRound + 1;
currJob!length >=currBitRound
false /* all bits of the sequence are received */
true /* there is one bit left of the current bit sequence */
RxBBFrame (arbitrating, rxFrame, currJob!length, currJob!id) to currJob!pid
TransComplete
disabled
nextBBTime := nextBBTime + dBurst; SET(nextBBTime, BurstIntervalT);
role
else /* node transmits its own bit sequence*/
REPEATER /* node lost contention or did not start to send ist own bit sequence */
currJob!frame(currBitRound-1)
1
else
TxShortBurst
rxFrame := rxFrame//mkstring(1)
wait4cmpl
listen

Fig. 7. ArbTransfer (excerpt 3)

5 Implementation Aspects for Time-Critical Systems

To implement time-critical systems specified in SDL, we have taken several measures in the SDL Virtual Machine (SVM) and the SDL Environment Framework (SEnF) (see Sect. 4.1). These measures support and extend the guidelines identified in Sect. 4.2 to improve timeliness and predictability of these systems. In addition, we have realized measures to speed up the execution times of SDL transitions and thereby reduce the execution delay of other fireable transitions.

Time-critical behavior in SDL systems can, e.g., be supported by adopting scenario-specific solutions, not necessarily limited to the specification itself. One example is the conceptual change from late to early timestamping for the CCA mechanism (see Sect. 4.4). As described in Sect. 3, the CCA mechanism and its accuracy are essential for the recognition and correct interpretation of black bursts on the medium. The moment a medium status change is detected, a hardware interrupt is triggered by the transceiver. This interrupt is handled directly by the transceiver driver included in the SEnF. We modified this driver in a way that the current hardware clock is read and the time value is assigned already during execution of the interrupt routine. This timestamp is then added as parameter to the CCA status signal. Since the time from interrupt creation to completion of the clock reading is constant, we can even correct the timestamp, leaving the hardware-dependent CCA jitter as the only (very small) source of inaccuracy.

Listings 1 and 2 are code excerpts of our SEnF transceiver driver. Listing 1 shows the interrupt routine where the timestamping occurs (line 5). It is triggered by a hardware interrupt when a medium status change is detected. The creation of the SDL signal `CC2420_CCA` during environment polling is presented in Listing 2. Here, the saved timestamp is added as second parameter to the signal, as shown in line 13. In summary, we have reduced unpredictable delays that affect the accuracy of black burst recognition to a minimum, while still using SDL as high-level design language. In particular, we can now guarantee a deterministic upper bound for the accuracy of black burst recognition, as the only remaining variable factor comes from hardware-dependent CCA jitter that has a known maximum value.

```
1  /* called when an interrupt on the CCA pin has occurred */ void
2  driver_cc2420_CCA_Interrupt () {
3    // get CCA status from transceiver and store timestamp from SVM
4      SEnF_CC2420_Data.actual_CCA = GPLR(CCA_PIN) & GPIO_BIT(CCA_PIN)
           ;
5    SEnF_CC2420_Data.actual_time = now();
6
7      envHasSignals = TRUE;
8  }
```

Listing 1. CCA interrupt routine from CC2420 transceiver driver

```
/* called by SVM during environment polling */
void send_CCA_signal() {
    // only if there's a change of the CCA signal between
    // last sending and now, a signal is send
    if (SEnF_CC2420_Data.actual_CCA != SEnF_CC2420_Data.last_CCA){

        // Macros to create and initialize SDL signal CC2420_CCA
        SENF_DECLARESIGNAL(CC2420_CCA, SignalIn)
        SENF_GETSDLSIGNAL(CC2420_CCA, SignalIn)

        // Assign signal parameters
        SDL_SIGNAL(SignalIn).Param1 = SEnF_CC2420_Data.actual_CCA;
        SDL_SIGNAL(SignalIn).Param2 = SEnF_CC2420_Data.actual_time;

        // Save current CCA status for comparison
        SEnF_CC2420_Data.last_CCA = SEnF_CC2420_Data.actual_CCA;

        // Send Signal to SDL
        SENF_SENDTOSDL(CC2420_CCA, SignalIn, 0)
    }
}
```

Listing 2. Creation of CCA signal during environment polling

Another aspect is the reduction of execution times, leading to reduced execution delays of SDL transitions. According to the SDL standard, variables of most SDL data types are copied by value, especially when used as signal parameters and exchanged between different SDL processes. This includes complex and memory intensive data types like `Octet_string` or Arrays. Although SDL-2000 offers a *copy by reference* semantics with *Object* data types, this semantics is limited to where a process instance contains both the output and input. The *copy by value* semantics can generate substantial unnecessary and avoidable overhead if the data is never modified by the involved SDL components and thus a single and SDL process-spanning instance of the data would be sufficient.

To efficiently implement the semantics of SDL data types, we are following two different approaches. First, we have specified and implemented new data types with *copy by reference* semantics as an alternative to existing data types. These can be either used if the data is never modified or if we explicitly want to work on the same data from multiple SDL processes cooperatively. Second, our SVM supports an optional *copy on write* mechanism on many existing SDL data types that is transparent for the SDL system and the generated code. *Copy on write* means that data is assigned by reference initially, not copied. The SVM keeps track of the number of references to this data. If a process performs a write access on multi-referenced data, the data is copied transparently and the modifications are only applied to the copy. Disadvantages of *copy on write* include additional overhead resulting from reference management and from checking the number of references before each write access. However, *copy on write* results in increased performance in many cases while preserving the SDL semantics. In our future work, we plan to evaluate to which degree existing SDL systems can benefit from *copy on write* semantics.

In our execution environment for motes, the SVM is implemented directly on top of the hardware platform and therefore has full control of system execution; no native operating system is used. In principle, this opens up possibilities for real-time scheduling of SDL transitions. However, we note that SDL does not provide many possibilities to control the order in which SDL transitions are fired. Basically, this is limited to priority inputs and to priorities of continuous signals, which place only certain local constraints on transition scheduling. The execution order of transitions belonging to different SDL processes can not be specified.

In time-critical systems, it is mandatory that (static and/or dynamic) priorities can be associated with processes. Furthermore, it must be possible to temporarily suspend processes in order to free resources. For instance, for a timely execution of time-critical transitions as in case of black burst handling, the CPU must not be overloaded. To solve the problem, global dynamic transition scheduling according to the specific needs of an SDL system is required.

To provide better control of the order in which SDL transitions are fired, several language extensions are perceivable. One step is the introduction of (static and/or dynamic) SDL process priorities, in order to enforce a global transition execution order. To reduce resource consumption during certain execution phases and thus have a more controllable, timely response time, these priorities could also be used to suspend processes. In other words, only those SDL processes with high priority remain active and can fire transitions during these periods. Extensive consideration of all related aspects is required before suitable language extensions of SDL can be proposed.

6 Conclusions

In this paper, we have presented the model-driven development of MAC-level, time-critical protocols for network-wide, collision-protected transfer of bit sequences in wireless networks with SDL-MDD. With these protocols, which support cooperative and arbitrating transfer based on encodings with black bursts, we have presented a solution to the problem of destructive frame collisions.

The contributions of the paper are twofold. First, we have shown how to formally specify low-level functionality and time-critical behavior, using the available constructs of SDL, a high-level design language. Secondly, we have extended the SDL Virtual Machine (SVM) and the SDL Environment Framework (SEnF) such that this high-level, time-critical design is supported automatically, using our SDL-MDD tool chain.

Instead of extending SDL, we have exploited the existing language constructs. In addition, we have identified a set of guidelines for time-critical specifications, which we have applied in our SDL design of MAC-level transfer protocols. Furthermore, we have extended the SVM and the SEnF, such that we have full execution control over the hardware platform. Thus, we were able to approximate time-critical behavior to a certain degree. Nevertheless, we would appreciate extensions of SDL to specify signal priority levels and process priorities.

In our future work, we will study the performance of low-level protocols with time-critical behavior. For this purpose, we are devising drivers and a real-time transition scheduler for the Imote2 hardware. Experiments will show whether the measures that we have introduced in this paper are sufficient. Also, based on experience gained, we will study further measures to support time-critical behavior.

References

1. Christmann, D., Gotzhein, R., Kuhn, T.: Multi-hop Clock Synchronization in Wireless Ad-Hoc Networks. In: Proceedings of Workshops on Mobile Ad-Hoc Networks (WMAN 2009), Kassel, Germany (2009)
2. International Telecommunications Union: Recommendation Z.100 (11/07), Specification and Description Language (SDL), http://www.itu.int/rec/T-REC-Z.100/en
3. Kuhn, T., Gotzhein, R., Webel, C.: Model-Driven Development with SDL - Process, Tools, and Experiences. In: Nierstrasz, O., Whittle, J., Harel, D., Reggio, G. (eds.) MoDELS 2006. LNCS, vol. 4199, pp. 83–97. Springer, Heidelberg (2006)
4. Gotzhein, R.: Model-driven with SDL – Improving the Quality of Networked Systems Development (Invited Paper). In: Proceedings of the 7th International Conference on New Technologies of Distributed Systems (NOTERE 2007), pp. 31–46 (2007)
5. Mohsen, A., Hofmann, R.: Efficient Voltage Scheduling and Energy-aware Co-synthesis for Real-time Embedded Systems. In: Srikanthan, T., Xue, J., Chang, C.-H. (eds.) ACSAC 2005. LNCS, vol. 3740, pp. 3–14. Springer, Heidelberg (2005)
6. IBM® Rational®, S.D.L.: Suite™, http://www.ibm.com/software/awdtools/sdlsuite/
7. PRAGMADEV: PragmaDev - real time development tools, http://www.pragmadev.com/
8. Fliege, I., Grammes, R., Weber, C.: ConTraST - A Configurable SDL Transpiler and Runtime Environment. In: Gotzhein, R., Reed, R. (eds.) SAM 2006. LNCS, vol. 4320, pp. 216–228. Springer, Heidelberg (2006)
9. Mitschele-Thiel, A.: Systems Engineering with SDL. Wiley, Chichester (2001)
10. Bræk, R., Haugen, Ø.: Engineering Real Time Systems. Prentice-Hall, Englewood Cliffs (1993)
11. Sanders, R.: Implementing from SDL. Telektronikk 4.2000, Languages for Telecommunication Applications. Telenor (2000)
12. Gotzhein, R., Kuhn, T.: Decentralized Tick Synchronization for Multi-Hop Medium Slotting in Wireless Ad Hoc Networks using Black Bursts. In: SECON, pp. 422–431. IEEE, Los Alamitos (2008)
13. Miller, J., Mukerji, J. (eds.): Object Management Group (OMG), MDA Guide Version 1.0.1, http://www.omg.org/docs/omg/03-06-01.pdf
14. International Telecommunications Union: Recommendation Z.120 (04/04), Message sequence chart (MSC), http://www.itu.int/rec/T-REC-Z.120/en
15. Gotzhein, R., Khendek, F., Schaible, P.: Micro Protocol Design - The SNMP Case Study. In: Sherratt, E. (ed.) SAM 2002. LNCS, vol. 2599, pp. 61–73. Springer, Heidelberg (2003)

Towards Model Based Development of Managed Networked Embedded Systems

El Arbi Aboussoror, Ileana Ober, and Michelle Sibilla

IRIT, Université de Toulouse, 118 Route de Narbonne
F -31062 Toulouse, France
{El-Arbi.Aboussoror,Ileana.Ober,Michelle.Sibilla}@irit.fr

Abstract. In this paper we focus on the use of models in the development of *managed networked embedded systems*. In this context, models offer a mechanism that allows the level of abstraction to be raised during development and more advanced functionalities such as adaptability to be targeted. In applications where management and configuration are central issues, there is a need for coordination between the development of the operational system and the development of the management system. In current practice, the two developments are often disconnected. This paper argues that the two developments should be done as much as possible in a collaborative way, and that integration of the developments is facilitated by the use of models in both developments. Finally, the paper explains how the model-based development in this area facilitates the addition of manageability and auto-configuration capabilities, which are premises for adaptability.

1 Introduction

The ever increasing use of embedded systems in our everyday environments, demands efficient mechanisms for developing these systems. This paper focuses on the development of networked embedded systems (NES). Embedded systems need to communicate and co-habit in a network, so their development has to meet the requirements of both embedded and networked systems. For networked systems the paper focuses in particular on the *management activities*. As detailed in [2], these are activities related to the operation, administration, maintenance and provisioning of networked systems. In most cases the management activities are done late in the development life-cycle, by independent developers teams, often after the operational part has been fully developed. This paper reports on work in progress that aims to define a generic development methodology for managed networked embedded systems. Our thesis is that the development of networked embedded systems should follow the model driven development (MDD) principles, in order to be able to abstract away, as much as possible, from platform dependent constraints and to explicitly express these constraints when they cannot be removed. The paper proposes that the design of the management part should be done as much as possible in parallel with the design of the operational part. The rest of this paper is organized as follows:

Section 2 overviews major issues existing in networked embedded systems design. Section 3 describes how MDD can be applied in networked embedded systems. There is a short overview of MDD (Sect. 3.1) and followed by exposing our approach (Sect. 3.2) and how it helps in *designing for manageability* (Sect. 3.3). These principles have been applied on the development of a case study that we use to validate our ideas (Sect. 3.4). The paper ends with a conclusion giving directions for future work.

2 Issues in Networked Embedded Systems Design

Nowadays, embedded systems are becoming more and more widespread in nearly all domains of our everyday life, such as household appliances, avionic applications, and medical devices. These systems often need to operate in a physically distributed manner, and therefore require tight connection between a set of embedded components on a network. Designers need to be able to manage a set of communicating systems. Such an aggregate system of NES raises new challenges on how to manage the distributed software artifacts and make them collaborate in an efficient way, how to meet the performance and QoS requirements, and how to design a dependable aggregate system of NES.

This paper gives a first step to answer these questions by proposing a base for a generic model-driven development methodology for designing NES. NES have the same design requirements as classical embedded systems:

– keep the manufacturing costs low;
– handle real-time constraints;
– reduce the resources usage (memory, power, etc.).

In addition to those requirements, NES should address some relevant management issues related to resource management, QoS and accounting.

With these additional requirements, new research challenges arise [3] on how to deal with dependability (timeliness, safety, security, availability), management (resources, network), and composability.

Management is a key activity in keeping the embedded systems operational and it has a significant influence on performance. Therefore handling management properly is a pre-requisite for meeting the QoS requirements. The management activities use available information about *processing elements*(e.g. router) and *dependencies* (e.g. a service relying on a web server) to perform their goals. This information is gathered in the management informational model. This approach is today widely used in several standards. For example:

– IETF SNMP [7] relies on the Management Information Base (MIB);
– the ITU-T M Series Recommendations especially the M.31xx Recommendations [9] provides either generic and specific network information models;
– the ISO System Management Overview (ISO/IEC 10040:1998) [8] introduces management information aspects.

All these standards make the information model a central part of the management artifacts. But only some of them are protocol independent. In the rest of this paper, the informational part of the management model is considered for these highlighted issues. And to remain generic, only the protocol independent standards will be considered. Currently in most cases the management informational model on which the system & network management is performed, is built late in the development life cycle, most often after the development of the system components. The deficiencies of this design approach are similar to those motivating the hardware and software co-design of embedded systems [6].

First, there is no representation of the entire system (operational and management parts) at design time: this representation is crucial for a performance analysis of the whole system. The impact of the management on the performance of a managed system [14] could be controlled if an early and complete representation of the system is available at design time. This issue is even more critical in the constrained context of NES. Second, the separation between operational system architecture and management system architecture leads to sub-optimal system design. Consequently the design of the management data collecting operations becomes tricky.

These problems are addressed by designers by considering the management requirements earlier.

3 MDD for Networked Embedded Systems Design

This section takes a look at the techniques used for NES design and at the place of models in this context.

3.1 Model Driven Development Principles

Today's software industry faces an increased interest in modeling and techniques aiming to raise the abstraction level in software development. One of the most relevant steps in this direction is the OMG initiative in promoting modeling through MDA/MDE [16]. Modeling is not a new issue: it is traditionally used in science, as a means to master the complexity. The key to success here is *abstraction*. In software development, languages such as SDL [10] or MSC [11], paved the way for the use of modeling. The adoption of the MDA/MDE technology at the level of individual projects is very much dependent on the existence of well-defined software development methodologies that clearly and completely identify the different development phases and their expected outputs; as well as the development actors and their responsibilities. In this context the arguments presented in [12] fifteen years ago are as fresh as ever. It is unrealistic to expect that enterprises that already have well established development methodologies are eager to abandon them. Therefore the target is a *generic development* process that can be customized according to the needs and specific situations of actual development process users. Several development method patterns exist in the literature, such as the waterfall model [18], the iterative model [15], and the spiral model [1].

The MDA advocates for a *Y-shaped* development process where the upper branches correspond to the platform independent development that leads to the PIM (Platform Independent Model) respectively to the platform dependent development that leads to the PSM (Platform Specific Model).

The actual system is obtained by merging the two (the lower part of the Y) in the actual model. Ideally this is (at least partially) done by automatic code generation.

3.2 Using Models in NES Design

The complexity of the management activities is increasing with the complexity of the managed network and systems. That raises new challenges, such as how to designate the managed components, what kind of management capabilities are needed, how to store and handle the management information.

Several initiatives to produce management protocol independent standards are addressing these issues by using an object oriented informational model, such as: the DMTF CIM [4] and the TM Forum Information Framework [21]. Almost all those approaches rely on introducing different abstraction levels. In such a setting, models are a promising candidate [20] to handle complexity. Our approach relies on the ISO standard *ISO/IEC 10040* [8] for designing the management system. According to this standard, in order to build the management model, four kinds of models have to be built:

- *The functional model*: states the administrator requirements and defines a set of operations to meet those requirements.
- *The informational model*: specifies all the information needed to perform the management operations.
- *The organizational model*: describes the management component's distribution pattern.
- *The communicational model*: describes the communication services offered to the management components.

Although the management activities are typically started soon after the end of the development activities, as far as we know, little is done to elaborate a development method integrating the design and development of the functional part with the management activities. Therefore in the current state of the art,the management operates on an already built (and often deployed) system: the system & network manager sees the operational system as a black box. This leads to a twofold activity of *discovering* some structural or behavioral properties of the system and *implementing* a managing application for the system. This activity is a central part of the whole management designer's mission, but it could be made easier if done in parallel with the operational system development. Our thesis is that the network management models and especially the *Informational model* should be built earlier, during the operational system analysis and design phase.

The initial hypothesis is that during the development a UML profile for management is used. This management profile can be the CIM Profile [5], but other management profiles can be used as well.

At design time the designers must take into account the artifacts specific to the targeted management activities. Thus, they should identify the *management model artifacts*. This is done, based on the management profile. At operational system design time, the entities meaningful for management can be identified and *flagged* as such. It is important to note that at this stage the only thing needed is to flag these entities.

For instance by stereotyping the model classes that will be meaningful for management. The reason for not going into more detail is that at this stage there is not enough information available to precisely classify these entities in terms of management. Moreover the operational system designer might not have the training needed to correctly classify these entities in terms of management.

In order to keep the operational system design independent from the management profile, a generic *management* stereotype will be used. This means that any management specific profile is imposed on the operational system design. Moreover, flagging the management specific entities offers a basis for automatically generating an initial management information model, based on the management profile chosen by the management experts. This generated model provides a starting point for the development of the management part and it covers most of the management specific *discovery* activity, while allowing the management expert to identify the actual nature of the discovered management entities and their relationships.

Concretely speaking, this means the management expert will stereotype the classes - flagged as management specific in the operational system development part - and will add the needed detail in terms of the chosen management profile.

This generated model will be refined by the management system designers, who will apply management stereotypes and patterns.

Once the management system model is achieved, design verification can start. All the verification activities (simulation, test & validation) could now take in their scope both the management system and the operational system. In the NES context this approach is pertinent because an operational system meeting the requirements does not lead necessary to an entire system (operational coupled with the management system) also meeting the requirements. To make things clear, imagine that a component is managed according to an agent-manager paradigm. The agent is embedded into the managed component, and in the NES context the agent is relying on the same power resource as the managed component. Thus the agent activity, if it is not taken into account during the power consumption analysis could lead to unpredicted runtime behavior.

Figure 1 schematizes the exchanges that are done between the system and management part developments in a MDD setting, where the development of the platform independent and platform dependent parts are separated as much as possible. On the platform independent branch of the management system development process, the steps to perform depend on the management development method adopted. The steps presented in Fig. 1 are those recommended by the ISO standard *ISO/IEC 10040* [8], and are those presented above. *A key aspect of our approach is the collaboration between the two Y-shaped processes.*

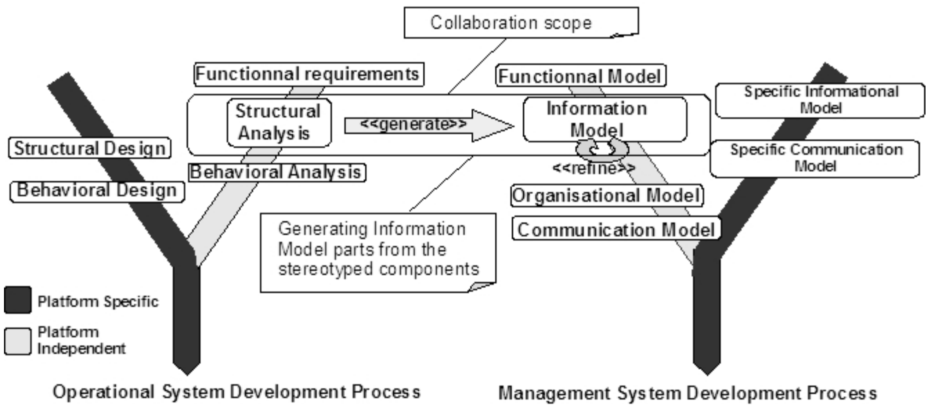


Fig. 1. Integrated development process

This collaboration between the operational and the management stakeholders roles is crucial and should result in *sharing and reusing* model elements.

The proposed integration has two main advantages. The first, is to make the development of the management application easier, by facilitating the identification of the management-relevant entities and generating a part of the corresponding data model and the basic management operations. The second advantage is to make the system under development easier to manage. In fact by taking into account the management requirements early in the development process, the designers are able to make available (or easy to access) some information from the system at management level. Therefore the algorithms used to perform the management tasks get less complicated. In fact management system designers are helping the operational system designers to *design for manageability.* This point is detailed in the following section.

3.3 Towards Highly Manageable Components

Manageability is the ability to gather information about the state of something and to control it [17]. If seen as a metric, manageability is the performance of the management system [14]. Manageability capabilities are necessary for a NES to remotely control the NES's configuration, to add maintenance and upgrade facilities, and to decrease costs of diagnosing runtime failures.

Based on these requirements we can define a *Highly Manageable Component* (HMC) as a component with three characteristics:

- HMC management is cost effective;
- HMC *efficiency* [14] is high (closed to 1), which means that the impact of the management system on the operational one is minimal;
- the HMC management system *productivity* is maximal.

This definition is performance metric independent. That means that there is a need of design effort to identify the adequate set of performance metrics [13].

The integrated development process (Fig. 1) introduced in this article facilitate the design and implementation of HMC. Our future work includes more investigation in this way.

3.4 Example

To test our approach, we applied it to a *fleeT tRAcking System* (TRAS), using UML for the system design and the UML Profile for DMTF Common Information Model (CIM) [5] for the management system design. The system is composed of several mobile entities that need to be tracked by a *supervisor*. Figure 2 gives an overview of the system, it shows their different components. Each vehicle of the fleet has an embedded *Tracer* component that sends its position to a server. The *Supervisor* can then connect to the server and track the vehicles. In this example the collaboration between the operational system designers and the management system designers occurs at the *Tracer* component design.

Figure 3 shows one of the *Tracer* subcomponents: the *TracerController*. This subcomponent is stereotyped as a *managed* component and its attributes are stereotyped with *managedSetting*. What we show in this figure is the *configuration* capability of this component. These stereotypes are applied by the system

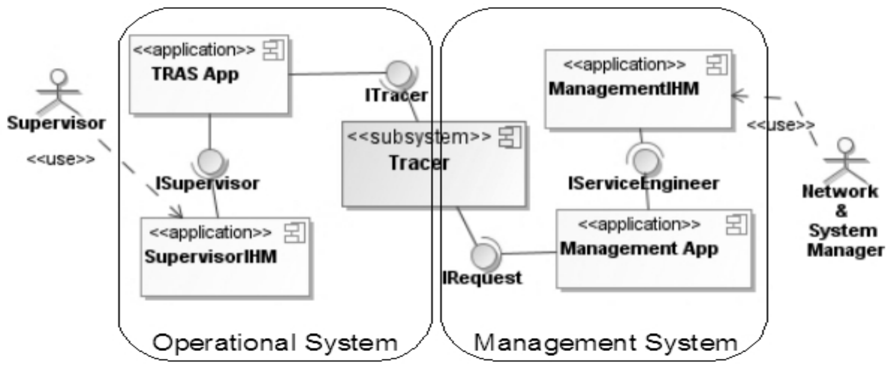


Fig. 2. Preliminary Architecture

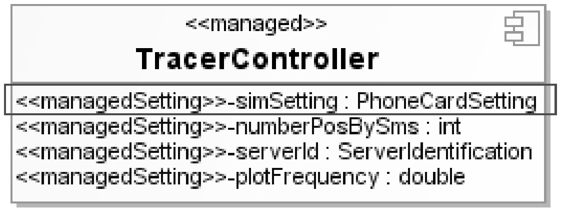

<<managed>>
TracerController
<<managedSetting>>-simSetting : PhoneCardSetting
<<managedSetting>>-numberPosBySms : int
<<managedSetting>>-serverId : ServerIdentification
<<managedSetting>>-plotFrequency : double

Fig. 3. A Managed Component of the Tracer Subsystem

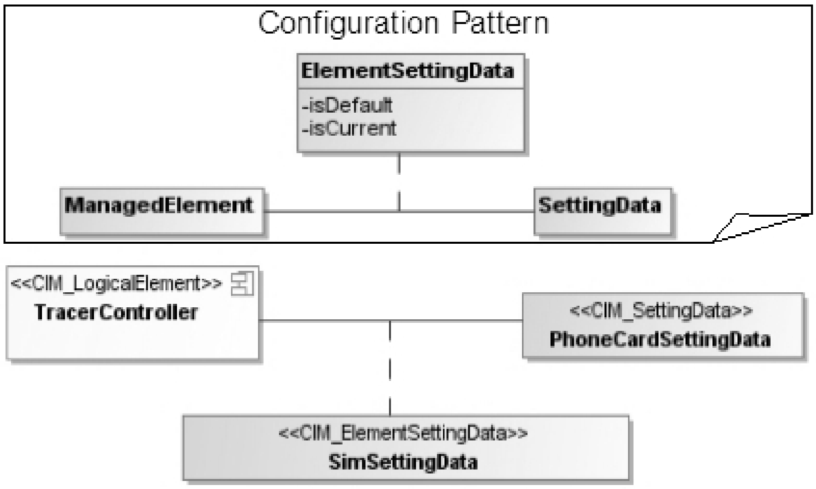


Fig. 4. Part of the *TracerController* Configuration Model (with the applied configuration pattern)

designer and they are specialized by the system management designers as shown in the Fig. 4. This part of the informational model shows the application of a configuration pattern from the CIM Core Specification [4] to the *TracerController* component. For specializing the management part, the management experts will use a management profile. In our case we use the CIM profile.

3.5 Discussion and Future Work

Our approach aims to bring the MDE benefits [20] into the management activities. Consequently the managed components can be captured; the management structural and behavioral data are implementation language independent. The advantage of our approach is the earlier collaboration introduced in the development process between the operational system designers and the system management designers. We claim that this integration will increase the quality of the management system. That means that it will improve the performance and the quality of the whole system at various levels, that range from system reliability, safety & security, QoS and maintainability. In particular it should lead to a better energy consumption control [3] since we can use a higher level energy management policy to optimize the system energy consumption (e.g. optimize the communication traffic and the networked embedded systems energy consumption in an integrated manner).

This list is not exhaustive, and we can also imagine that this integration is a first step to auto-configurable component design. That allows the introduction of auto-configuration behavior in the embedded components earlier and more easily especially if the system behavior modeling covers the management part as well [19]. The auto-configuration is formalized by the management design stakeholders, and their earlier participation in the development process provides the component with more autonomic behavior.

4 Conclusion

This paper advocates the use of MDD techniques in the development of Networked Embedded Systems. We argue that the management design activities should come early in the development phases. This makes sure the information needed for management is available at the right moment. This also offers the premises for auto-adaptation. Indeed, the auto-adaptation can only be made if the (meta-)information from the operational part of the system is available. By designing the management part in parallel with the operational part, it is better identified what information is needed and what could be used in an autonomous setting. We intend to continue the integration of operational part development with the management part development, in the context of a model based development. In particular, we intend to look closer for relationships that may exist between the models developed, with the aim of enabling the automatic exchange of information between the two.

References

1. Boehm, B.W.: A spiral model of software development and enhancement. IEEE Computer 21(5), 61–72 (1988)
2. Clemm, A.: Network Management Fundamentals. Pearson Education, London (2007)
3. National Research Council: Embedded Everywhere. National Academy Press (2001)
4. DMTF: Common Information Model (CIM) Standards, http://www.dmtf.org/standards/cim
5. DMTF: UML Profile for CIM, http://www.dmtf.org/standards/published_documents/DSP0219.pdf
6. The Donald O. Pederson Center for Electronic Systems Design: A framework for hardware-software co-design of embedded systems, http://embedded.eecs.berkeley.edu/Research/hsc/abstract.html.
7. IETF: RFC 1157: A Simple Network Management Protocol, http://www.ietf.org/rfc/rfc1157.txt
8. ISO: ISO/IEC 10040:1998 Systems Management Overview, http://www.iso.org/iso/catalogue_detail.htm?csnumber=24406
9. International Telecommunications Union: M Series Recommendations, Telecommunication management, including TMN and network maintenance, http://www.itu.int/rec/T-REC-M
10. International Telecommunications Union: Recommendation Z.100 (11/07), Specification and Description Language (SDL), http://www.itu.int/rec/T-REC-Z.100/en
11. International Telecommunications Union: Recommendation Z.120 (04/04), Message sequence chart (MSC), http://www.itu.int/rec/T-REC-Z.120/en

12. Jackson, M.: Problems, methods and specialization. IEEE Software 11(6), 57–62 (1994)
13. Jain, R.: The Art of Computer Systems Performance Analysis. Wiley, India (2008)
14. Lahmadi, A., Andrey, L., Festor, O.: On the impact of management on the performance of a managed system: A jmx-based management case study. In: Schönwälder, J., Serrat, J. (eds.) DSOM 2005. LNCS, vol. 3775, pp. 24–35. Springer, Heidelberg (2005)
15. Larman, C., Basili, V.R.: Iterative and incremental development: A brief history. Computer 36(6), 47–56 (2003)
16. Miller, J., Mukerji, J. (eds.) Object Management Group (OMG), MDA Guide Version 1.0.1, http://www.omg.org/docs/omg/03-06-01.pdf
17. The Open Group: The Open Group architecture framework TOGAF, http://www.opengroup.org/togaf/
18. Royce, W.W.: Managing the development of large software systems: concepts and techniques. In: ICSE 1987: Proceedings of the 9th international conference on Software Engineering, pp. 328–338. IEEE Computer Society Press, Los Alamitos (1987)
19. Sibilla, M., Barros de Sales, A., Broisin, J., Vidal, P., Jocteur-Monrozier, F.: Behaviour modelling: a contribution to CIM, http://www.dmtf.org/education/academicalliance/sibilla_2004.pdf
20. Terrier, F., Gérard, S.: MDE benefits for distributed, real time and embedded systems. In: DIPES. IFIP, vol. 225, pp. 15–24. Springer, Boston (2006)
21. TM Forum: Information Framework (SID), http://www.tmforum.org/InformationFramework/1684/home.html

Towards an Integrated Quality Assessment and Improvement Approach for UML Models

Akhtar Ali Jalbani[1], Jens Grabowski[1],
Helmut Neukirchen[2], and Benjamin Zeiss[1]

[1] Institute for Computer Science, University of Göttingen
Goldschmidt Str. 7, 37077 Göttingen, Germany
{ajalbani,grabowski,zeiss}@informatik.uni-goettingen.de
[2] Faculty of Industrial Engineering, Mechanical Engineering and Computer Science
University of Iceland, Dunhagi 5, 107 Reykjavík, Iceland
helmut@hi.is

Abstract. Models defined using the *Unified Modeling Language* (UML) are nowadays common parts of software documentations, specifications and sometimes even implementations. However, there is a broad variety of how UML is used. Reasons can be found, for example, in the lack of generally accepted modeling norms and guidelines, the semi-formal semantics of UML, or the complexity of the language. In practice, these factors inevitably lead to quality problems in UML models that need to be addressed. We investigate and discuss existing work in the field of quality assessment and improvement of UML models and present how we envision an integrated approach to quality assessment and improvement of UML models. We assess a model with a *Factor-Criteria-Metrics* (FCM) based quality model, detect issues by finding smells and violated metric thresholds in UML models, and improve UML models by applying refactorings using model-to-model transformations.

1 Introduction

Quality control for a software development process requires ongoing quality assurance measures for all artifacts produced during the development process. Assessing the quality of artifacts produced in early phases of a process, such as requirement or design specifications, is critical since a change in these specifications often implies change in documents, specifications, or code (during later development phases) as well. The *Unified Modeling Language* (UML) from the *Object Management Group* (OMG) has been widely adopted as a common modeling language for the creation of requirements and design specifications. In later stages of development, UML models may also serve as a base for code and test generation. Unfortunately, there is a wide variety of how UML is used in practice. This is often due to the lack of generally accepted modeling norms and guidelines, the semi-formal semantics of UML and the complexity of UML as a whole. In practice, these factors inevitably lead to quality problems in UML models that need to be addressed. Therefore, continuous tool-supported quality assurance and quality improvement measures, throughout the whole development process, are required. Based on our experience [1,2] with the quality engineering of large test specifications written in the standardized *Testing and Test Control Notation* (TTCN-3), we started to investigate the possibility of using similar techniques for the quality engineering of UML models. Our quality engineering approach for TTCN-3 specifications is based on:

– a quality model for test specifications that defines the main quality characteristics of a test specification,
– test metrics to assess the quality characteristics,
– smell detection for identifying problematic locations in the test code using pattern-based analysis and metric thresholds, and
– refactoring for the improvement of those problematic locations.

While TTCN-3 from its appearance is comparable to a typical general purpose programming language like C or Java, the challenges to adopt this approach for UML are numerous. As already mentioned, the usage and actual knowledge about UML is very diverse. It is not unusual that people mistake UML as a standardized way to draw diagrams, rather than understanding it as a modelling language, that uses diagrams for partial presentations of the model. This variety of how UML is perceived differently by the people using it, eventually has an effect on what UML specifications look like, how they can be used later on, and also how quality engineering must be implemented for UML specifications. The contribution of this paper is a survey on the topics quality models, metrics, bad smells, and refactoring for UML models. The papers investigated in the survey are selected to cover those topics that are related to our proposed approach for UML quality assurance.

This paper is structured as follows: in Sect. 2, we introduce the foundations of this paper: that is the basics of UML, quality models, metrics, bad smells, and refactoring. Sections 3–6 present existing work on the respective topics. Section 7 outlines current tool support for metrics, smell detection, and refactoring for UML. In Sect. 8, we present our ideas for an integrated quality engineering approach for UML. We conclude with Sect. 9, where we summarize our progress towards the realization of our ideas and we discuss what topics still need to be addressed.

2 Foundations

In this section, we briefly provide the foundations of the topics that are relevant for this paper — the *Unified Modeling Language* (UML), quality models, metrics, bad smells, and refactoring. Section 2.1 is of particular importance as it also attempts to address common misunderstandings and misconceptions about UML: what the UML architecture is about and what kinds of notations exist for UML models.

2.1 The *Unified Modeling Language* (UML)

We assume basic knowledge of UML and therefore we concentrate on facts that may not be immediately apparent to everyone who has not dealt with UML as a modelling language. The UML architecture is composed of four layers (Fig. 1). The M3 layer, the foundation of UML, is called the *Meta-Object Facility* (MOF) [3]. OMG itself describes MOF as a metadata management framework and metadata services. In essence, MOF is a language that is used to model itself as well as other models or metamodels. In the context of UML, the most prominent use of MOF is the definition of the UML metamodel (the M2 layer). MOF can be considered a meta-metamodel in this case. A distinction is made between *Essential MOF* (EMOF) and *Complete MOF* (CMOF). As the names suggest, EMOF is a slimmed down version, a subset of CMOF. MOF is used to specify the UML metamodel that consists of the *Infrastructure* [4] and *Superstructure* [5] standards. These standards define the abstract syntax of the language: that is basic UML modeling concepts, attributes, relationships, as well as the semantics of each modeling concept. In the language definition, the cohesion between the Infrastructure definition and MOF is more complex, as MOF is again defined using UML. The M1 layer is again an instance of the M2 layer. On the M1 layer, we find those models that we typically create for requirements or design specifications. The instance of a UML model is then finally found on the M0 layer, which describes instantiated objects.

The UML models we deal with everyday are typically the ones found on the M1 layer — we create instances of the UML metamodel. One common way to create such a model is to use the graphical notation provided in the UML Superstructure standard. However, it is crucial to understand that a UML model and a UML diagram are different things. The diagrams defined in the UML Superstructure are essentially guidelines and do not represent a specific formal notation. This results in ambiguities and variations between tools. It is easily possible to draw a set of diagrams in the UML notation on paper. However, on paper these cannot be validated, transformed, or even used for code generation.

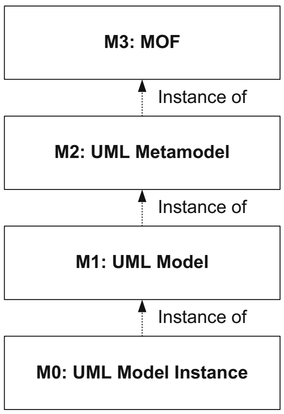
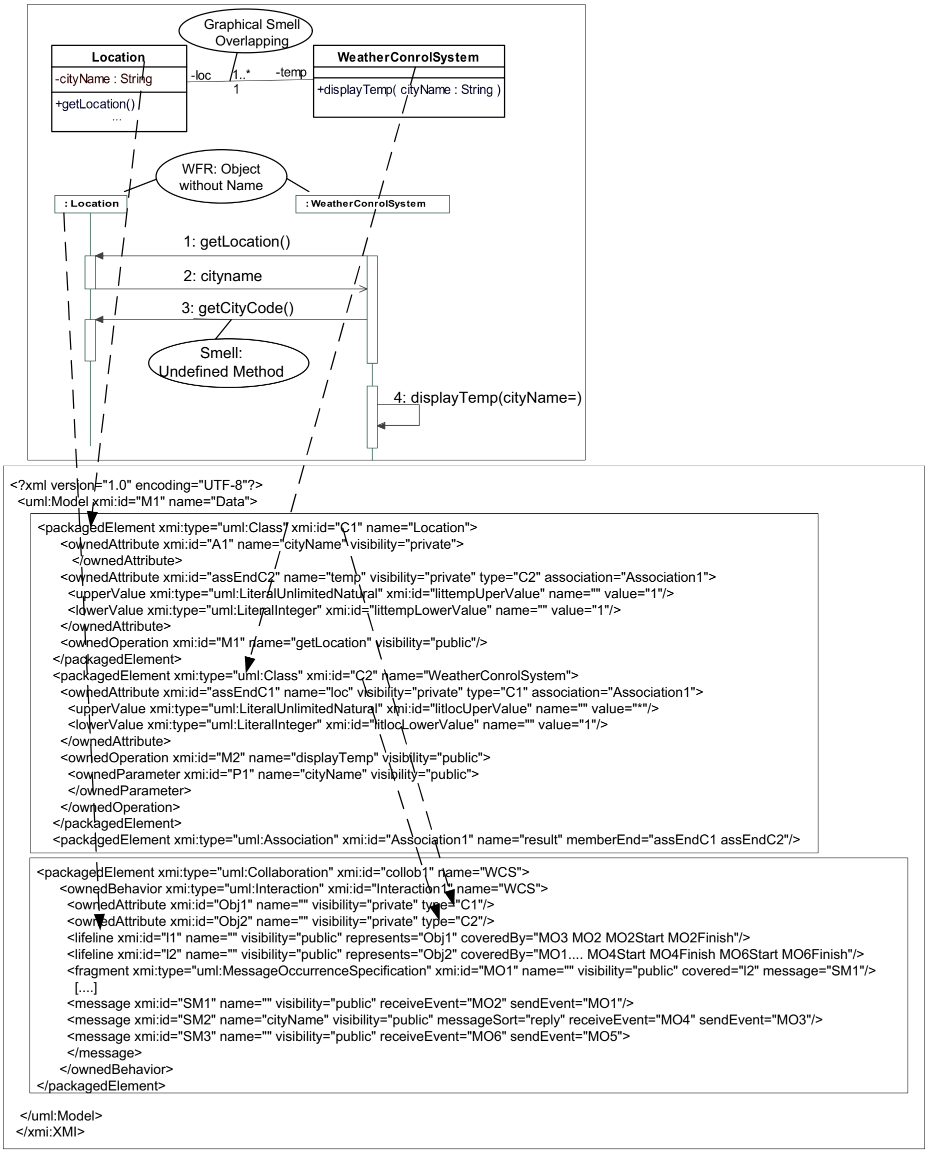


Fig. 1. The UML Architecture



```
<?xml version="1.0" encoding="UTF-8"?>
 <uml:Model xmi:id="M1" name="Data">
   <packagedElement xmi:type="uml:Class" xmi:id="C1" name="Location">
      <ownedAttribute xmi:id="A1" name="cityName" visibility="private">
        </ownedAttribute>
      <ownedAttribute xmi:id="assEndC2" name="temp" visibility="private" type="C2" association="Association1">
        <upperValue xmi:type="uml:LiteralUnlimitedNatural" xmi:id="littempUperValue" name="" value="1"/>
        <lowerValue xmi:type="uml:LiteralInteger" xmi:id="littempLowerValue" name="" value="1"/>
      </ownedAttribute>
      <ownedOperation xmi:id="M1" name="getLocation" visibility="public"/>
    </packagedElement>
    <packagedElement xmi:type="uml:Class" xmi:id="C2" name="WeatherConrolSystem">
      <ownedAttribute xmi:id="assEndC1" name="loc" visibility="private" type="C1" association="Association1">
        <upperValue xmi:type="uml:LiteralUnlimitedNatural" xmi:id="litlocUperValue" name="" value="*"/>
        <lowerValue xmi:type="uml:LiteralInteger" xmi:id="litlocLowerValue" name="" value="1"/>
      </ownedAttribute>
      <ownedOperation xmi:id="M2" name="displayTemp" visibility="public">
        <ownedParameter xmi:id="P1" name="cityName" visibility="public">
        </ownedParameter>
      </ownedOperation>
    </packagedElement>
    <packagedElement xmi:type="uml:Association" xmi:id="Association1" name="result" memberEnd="assEndC1 assEndC2"/>
   <packagedElement xmi:type="uml:Collaboration" xmi:id="collob1" name="WCS">
      <ownedBehavior xmi:type="uml:Interaction" xmi:id="Interaction1" name="WCS">
        <ownedAttribute xmi:id="Obj1" name="" visibility="private" type="C1"/>
        <ownedAttribute xmi:id="Obj2" name="" visibility="private" type="C2"/>
        <lifeline xmi:id="l1" name="" visibility="public" represents="Obj1" coveredBy="MO3 MO2 MO2Start MO2Finish"/>
        <lifeline xmi:id="l2" name="" visibility="public" represents="Obj2" coveredBy="MO1.... MO4Start MO4Finish MO6Start MO6Finish"/>
        <fragment xmi:type="uml:MessageOccurrenceSpecification" xmi:id="MO1" name="" visibility="public" covered="l2" message="SM1"/>
          [....]
        <message xmi:id="SM1" name="" visibility="public" receiveEvent="MO2" sendEvent="MO1"/>
        <message xmi:id="SM2" name="cityName" visibility="public" messageSort="reply" receiveEvent="MO4" sendEvent="MO3"/>
        <message xmi:id="SM3" name="" visibility="public" receiveEvent="MO6" sendEvent="MO5">
        </message>
      </ownedBehavior>
   </packagedElement>

  </uml:Model>
 </xmi:XMI>
```

Fig. 2. Graphical and XMI Representation of a UML Model

Even if we transfer our diagrams as they are into digital form, they are missing important pieces of information that are not part of the diagrams such as how the diagrams relate to each other and where the definitions to model references can be found. If the graphical notation is used to create a UML model (i.e., by using a UML tool), each diagram represents only a partial view of the complete model. Thus, a UML model may be described by multiple diagrams or no diagram at all — a UML model may still contain all elements we know from the commonly used graphical notation without including a single diagram. However, there is no common and unified notation that can represent a UML model completely, but attempts to solve this problem exist, e.g, TextUML [6]. One way, although not entirely human-readable, to represent a complete UML model is the *XML Metadata Interchange* (XMI) format [7] which is, however, an exchange format rather than a useful notation for modeling.

To illustrate the difference between a model and diagrams, we present a simple specification of a weather information system in Fig. 2. At the top of the figure, we have the graphical notation of a UML model consisting of a class and a sequence diagram. At the bottom part of the figure, we present the XMI notation of the same model. The figure illustrates two things. First, a complete model can represent multiple diagrams and vice versa — multiple diagrams may be part of a single UML model. In this case, the model contains the definitions from the class diagram and the sequence diagram. Second, the XMI representation explicitly references the previously defined UML classes. Such an explicit reference is not possible when we deal with diagrams in UML notation (that are created using pencil and paper or a diagramming tool) rather than UML models. Finally, it is necessary to mention the *Object Constraint Language* (OCL) [8]. OCL is a declarative language used to express constraints (preconditions, postconditions, or invariants) on UML models and is based on first-order predicate logic. Although not its intended use, it is also possible to use OCL as a query language by evaluating result sets of OCL expressions.

2.2 Software Quality and Quality Models

Software quality refers to all attributes of a software product that show the appropriateness of the product to fulfill its requirements. For a software product, Fenton et al. [9] distinguish between attributes of *processes*, *resources*, and *products*. For each class, *internal* and *external attributes* can be distinguished. External attributes refer to how a process, a resource, or a product relates to its environment. Internal attributes, on the other hand, are properties of a process, a resource, or a product on its own, separate from any interactions with its environment. Hence, the assessment of external attributes of a product, the so-called *external quality*, requires the execution of the product, whereas usually static analysis is used for the assessment of its internal attributes, the so-called *internal quality*. Since this article treats quality characteristics for UML models, which are products that do not need to be executable, only internal quality is considered in the following.

Quality models are used to assess software quality. Our work concentrates on hierarchical *Factor-Criteria-Metrics quality model*s (FCM-models). Prominent examples for FCM-models are the quality model developed by McCall et al. (McCall-model) [10] and the ISO/IEC standard 9126 (ISO/IEC 9126-model) [11].

The highest level of the McCall-model are the three uses: *operation*, *transition* and *maintenance*. The operation use refers to quality characteristics that concern the product when it is being executed, i.e., its external quality. The

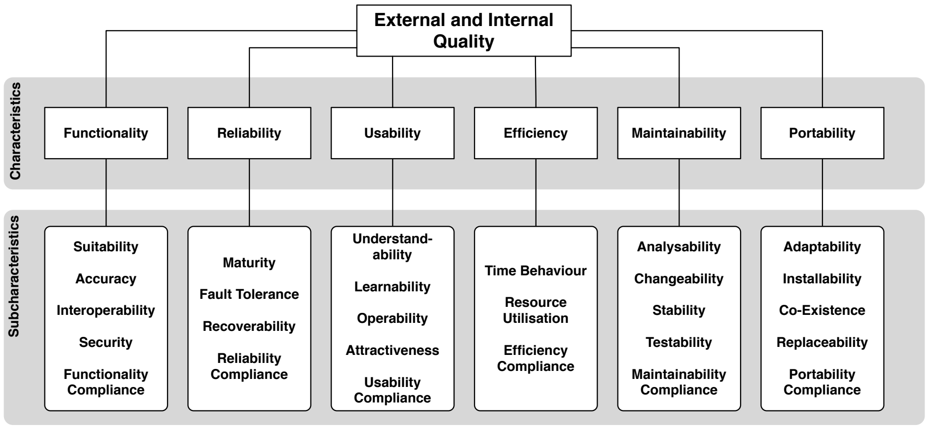


Fig. 3. ISO/IEC 9126 Quality Model for Internal and External Quality

transition use combines quality characteristics that concern the product when it is moved to another environment, and the maintenance use focuses on quality characteristics that concern the product when it is changed. As indicated by the abbreviation FCM, on the second, third and fourth level, the McCall model defines *factors*, *criteria* and *metrics*. A factor defines a high-level quality criterion such as efficiency. On the next lower level, criteria for judging factors are defined. For example, criteria for the factor efficiency are storage and execution efficiency. Metrics are then used to assess criteria, e.g., storage efficiency may be assessed by calculating the ratio between allocated and used storage.

The ISO/IEC 9126-model defines no uses, but distinguishes between internal quality, external quality and quality-in-use. The quality ISO/IEC 9126-model is a generic quality model that covers internal and external quality in one abstract model (Fig. 3). The model for *quality-in-use* is similar to the operation use of the McCall model. However, quality-in-use and external quality are out of the scope of this paper and therefore not discussed any further. In the ISO/IEC 9126-model, factors are called *characteristics* and criteria are called *subcharacteristics.*

2.3 Software Metrics

Fenton et al. structured internal product metrics (metrics that measure internal quality) into *size* and *structural* metrics [9]. Size metrics measure properties of the number of usage of programming or specification language constructs: for example, the *number of source statements.* Structural metrics analyze the structure of a program or specification. Popular examples of structural metrics are complexity metrics based on control flow or coupling metrics.

To make sure that reasonable metrics for quality assessment are chosen, Basili et al. suggest the *Goal, Question and Metrics* (GQM) approach [12]: First, the goals which shall be achieved (such as improved maintainability) must be defined. Then, for each goal, a set of meaningful questions that characterize a goal is derived. The answers to these questions determine whether a goal has been met or not. Finally, one or more metrics are defined to gather quantitative data which can provide answers to each question.[1]

2.4 Smells

The metaphor of "*bad smells in code*" has been coined by Beck and Fowler [13]. They define smells as "*certain structures in the code that suggest (sometimes they scream for) the possibility of refactoring*". According to this definition, defects with respect to program logic, syntax, or static semantics are not smells, since these defects cannot be removed by a refactoring. A refactoring only improves internal structure, but does not change observable behaviour.

Beck and Fowler present smells for Java source code. They describe their smells using unstructured English text. A well-known smell is *Duplicated Code.* Code duplication affects (in particular) the *changeability* quality subcharacteristic in the ISO/IEC 9126-model: if code that is duplicated needs to be modified, it usually needs to be changed in all duplicated locations. Smells provide only hints: whether the occurrence of an instance of a certain smell in a source code is considered as a sign of low quality may depend on preferences and the context of a project. For the same reason, a list containing code structures that are considered smells is never complete and may also vary from project to project and from domain to domain [14].

The notions of metrics and smells are not disjoint: each smell can be turned into a metric by counting the occurrences of a smell, and often, a metric can be used to locate a smell. The latter is the case, for example, when a long function is expressed by a metric that counts the lines of code of this function and a threshold is violated. However, the above smell of duplicated code and other pathological structures in code require a pattern-based detection approach and cannot be identified by using metrics alone.

2.5 Refactoring

Refactoring is defined as "*a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior*" [13]. This means that refactoring is a remedy against software aging [15]. While refactoring can be regarded as cleaning up source code, it is more systematical and thus less error prone than arbitrary code clean-up, because each refactoring provides a checklist of small and simple transformation steps, which are often automated by tools.

The essence of most refactorings is independent from a specific programming language. However, a number of refactorings make use of particular constructs of a programming language, or of a programming paradigm in general, and are thus only applicable to source code written in that language.

[1] The GQM approach can also be used to define individual FCM quality models as goals are similar to factors and questions similar to criteria.

3 Quality Models for UML

A surprisingly small number of researchers have addressed the problem of quality assessment for UML models. The comprehensive work in this area has been done by Lange and Chaudron [16,17]. In [17], they discuss the difference between source code and UML models and highlight the particularities of UML models. As a consequence, a special quality model for UML has been developed (in the following called Lange-Chaudron-model). An overall view of the model is given in Fig. 4.

Like the model developed by McCall, the Lange-Chaudron-model is a hierarchical model with four levels. On the highest level, the Lange-Chaudron-model defines the two uses *maintenance* and *development*. The maintenance use is taken from the McCall model. The other two uses from McCall (*operation* and *transition*) are not relevant for the quality of UML models. The operation use is related to external quality attributes and the transition use is not related to the modeling phase and design phase of development in which UML is used. The development use combines quality characteristics of a product and its artifacts in phases before the product is finished. The second level of the Lange-Chaudron-model defines the purposes of modeling. For example, the purpose *Testing* indicates that the model is used for test generation and the purpose *Code Generation* denotes a usage for automatic code generation. The third level of the Lange-Chaudron-model identifies the characteristics of the purposes. The meaning of most characteristics in Fig. 4 is straightforward. For example, the characteristic *complexity* measures the effort required to understand a model or a system.

Two special characteristics of the Lange-Chaudron-model are *aesthetics* and *balance*. The quality of the graphical diagrams is addressed by the aesthetics characteristic only. Aesthetics is defined by the extent that the graphical layout of a model or system enables ease of understanding of the described system. Lange and Chaudron define balance as the extent that all parts of a system are described at an equal degree. All characteristics are included in the balance characteristic with the same weight. This has been criticized by Mohagheghi and Aagedal [18], because the assessment of the balance characteristic requires the evaluation of all metrics and rules defined in the fourth level, i.e., it is not a good abstraction. In [18], it is proposed to shift balance to the purpose level and to assess balance by using the characteristics completeness, conciseness, modularity, and self-descriptiveness.

The fourth level of the Lange-Chaudron-model (not shown in Fig. 4) defines metrics and rules for the assessment of the characteristics. We discuss this part of Lange-Chaudron-model in Sects. 4 and 5. Lange and Chaudron underpinned their work with industrial case studies. They showed the applicability of their approach by interviewing representatives of project teams, analyzing UML models, and giving feedback to project teams. A quality model for design documentation in model-centric domains has been developed by Pareto and Boquist [19]. The background of this work is experience with the *Rational Unified Process* (RUP) as model-centric software development process. Even though UML is an essential

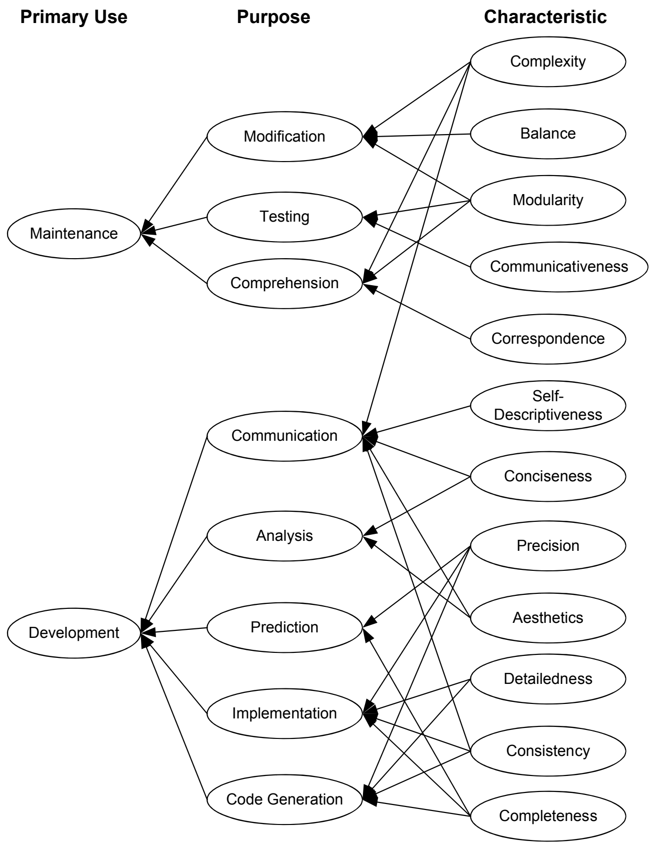


Fig. 4. Lange-Chaudron Quality Model

part of RUP, all kinds of artifacts on the abstraction levels between requirements specification and code are considered relevant. For the development of the quality model, Pareto and Boquist interviewed and discussed with designers, process engineers, line managers and architects. From these interviews and discussions, 22 quality attributes were identified and structured into six groups. Each group identified one quality characteristic. As the quality model is related to RUP also quality aspects for management are covered. However, they stop with the identification of quality attributes and quality characteristics. No means for the assessment of quality attributes and characteristics are provided.

4 Metrics for UML

In current research, a large number new software metrics are defined. Many metrics are based and calculated on grammatical structures such as *Abstract Syntax Tree*s (ASTs). A UML model is also based on a specific structure: the UML metamodel.

However, numerous proposals are based on informal metrics definitions that often respect only diagrams: the graphical representation of UML with its partial views. In the following sections, we will present noteworthy literature on UML metrics. We differentiate between metrics that are based on the actual UML model, and metrics that are solely based on the graphical notation: graphical metrics.

4.1 Model Metrics

Lange [20] uses metrics and rules (metrics with a binary result) and relates them to quality characteristics of his quality model (see Sect. 3) to assess the quality of a UML model. He reuses the most widely known metrics such as the metric suite from Chidamber and Kemerer [21] and describes them informally. He stresses that his list is by no means complete. Kim and Boldyreff [22] propose 27 metrics for UML that are supposed to predict characteristics at earlier stages in the software lifecycle. The metrics are defined informally and no relationship between the UML model quality and the metrics is established. Baroni et al. [23] propose to use OCL to describe UML metrics in a formal way in order to avoid ambiguities due to descriptions in natural language. By using several samples of different complexity, they demonstrate that OCL is a well suited formalism for defining UML metrics and that it is easier to understand than formulas using custom built mathematical frameworks. McQuillan and Power [24] extended this approach and use OCL to calculate coupling and cohesion metrics, as well as the metrics from the Chidamber and Kemerer metric suite [21]. They argue, however, that a metrics specific metamodel is a more generic solution than defining metrics directly over the UML metamodel. Furthermore, they demonstrate how to automatically generate test data and metamodel instances. Another interesting way to formalize metrics is proposed by El-Wakil et al. [25]. They propose to define metrics using *XQuery* over the XMI representation of the UML model under analysis. They argue that using XQuery to express metrics eases tool building. Also, they claim that metric libraries specified in XQuery are easy to extend and provide a proof-of-concept implementation.

4.2 Graphical Metrics

Graphical metrics for UML are not covered very well in the literature despite the fact that the quantification of visual elements can be an important part to assess the quality of a graphical layout. However, it seems that layouting itself draws more attention in research than the assessment of a layout by numbers. Kiewkanya and Muenchaisri [26] performed an experiment in which they evaluated whether metrics quantifying aesthetic aspects of class and sequence diagrams influence the maintainability of UML models. For the measurements, they selected aesthetic indicators that have been proposed by Purchase [27], Eichelberger [28], and others. Such aesthetic indicators are, for example, the maximum number of bends on the edges, the standard deviation of edge lengths, or the total numbers of edges fixed to an orthogonal grid divided by the total number of edges. Their conclusion is that aesthetic metrics can indeed be indicators for the maintainability of class and sequence diagrams. Gronback [29] provides a general catalog of UML metrics to detect deviations from best practices. Some of them are derived from style guidelines provided by Ambler [30]. He suggests generic diagram metrics such as "number of colors on diagram" or diagram-specific metrics such as "depth of inheritance hierarchy" (for class diagrams) and even provides minimum and maximum thresholds for his metrics. The metrics presented by Gronback, however, mix graphical properties with properties that are part of the UML model.

5 Smells for UML

As discussed earlier, UML models do not have a standardized textual notation like typical general purpose programming languages. However, bad smell analysis in source code is rarely executed directly on the textual notation. An abstract grammatical representation of the notation, the AST, can in fact be regarded as a model for the textual notation of the programming language that is subject of the analysis. Analyzing UML models is therefore not that much different than analyzing an AST. However, the underlying abstract syntax is more complex. In the following section, we present related work that deals with bad smells in UML models. We differentiate between model smells and graphical smells. With model smells, we regard design flaws or possible defects that we find by analyzing the UML model (independently from any diagrams) such as possible inconsistencies, ambiguities, or constructs that complicate maintenance. Graphical smells, on the other hand, are related to the graphical notation of UML. They primarily concern the understandability aspect of the diagram. For example, diagrams with overlapping or crossing elements are harder to understand than diagrams with elements that are laid out with aesthetic aspects in mind.

5.1 Model Smells

Lange [16] — with his goal to improve the overall quality of UML models — discusses that undetected defects can cause large problems in later development stages and identifies generic UML defects such as the number of messages in a sequence diagram that do not correspond to a method in a defined class diagram. The presented smells were identified by discussions with industrial partners and by performing case studies. He assumes that a set of UML diagrams defines a system as a whole and that those diagrams have consistency relationships between each other. The defects partially overlap with the well-formedness rules and are related in their scope, but are described informally, without a relationship to the abstract syntax of UML. Astels [31] presents UML smell detection in the context of UML refactoring. With smell detection, he locates where to refactor and which refactoring is suggested. He argues that the visual presentation of UML makes smell structures more evident and presents exemplarily what classical bad smells from Fowler [32] (e.g., lazy class or middle man) look like in the graphical notation. His own statement is that his list is by no means complete. His work is described informally in the visual notation of UML.

5.2 Graphical Smells

Graphical smells concern the graphical notation of UML models excluding problems that are of logical nature or that may introduce issues in efficiency or maintenance. Therefore, the main aspect of graphical smells is how model elements are laid out and what elements are represented by the diagrams. Ambler [30] provides more than 300 guidelines for all UML diagram types that primarily concern the graphical notation. The violations of these guidelines can be considered as graphical smells. As an example, a guideline with the aim to improve the understandability of a diagram is to split large diagrams with a high number of elements into multiple smaller diagrams, where no diagram must have more than nine elements. Purchase et. al [27] have studied graphical layout aesthetics in class and collaboration diagrams. By performing a case study where they questioned persons to investigate their subjective preferences, they conclude that there are certain common aesthetic properties that seem to be unfavorable. Among these properties are, for example, arc crossings, or orthogonality (for class diagrams). From their results, they derive that the aesthetics of graph layouts is dependent on the domain, i.e., properties that are important for one diagram type may not be important for another one.

6 Refactorings for UML

UML refactoring is an emerging research topic that can already be considered as important as classical source-code refactoring. We again differentiate between model refactorings, i.e., semantically preserving model changes and graphical refactorings that improve the aesthetics of UML diagrams.

6.1 Model Refactorings

Astels [31] presents UML smells in class and sequence diagrams and describes a number of Fowler refactorings that are applicable to UML. His refactoring descriptions are based on UML diagrams and are informal. His examples are intended to motivate that UML refactoring is applicable in the context of agile development processes. France and Bieman [33] want to avoid uncontrolled change and increased evolution costs of a system due to deteriorating structure and system quality by introducing a goal-directed, cyclic process for object-oriented software when object-oriented models, such as UML models, are transformed and evaluated in each cycle. For the model transformation, they explicitly mention model refactoring to enhance quality attributes of the model that should be realized using patterns involving roles, i.e., each participant in the pattern plays a certain role with specific properties within the pattern description. A formal method for pattern-based transformation with role models does not exist yet. Sunyé et al. [34] propose refactorings for class diagrams and state charts to make software easier to extend and maintain. Using pre and post conditions expressed in OCL, they ensure that transformation preserve behavioral properties. However, they describe the refactoring mechanics informally. Porres [35] presents how to describe and execute UML refactorings using a rule-based transformation formalism and he argues that an update-based mapping mechanism that modifies a model in place is more efficient for describing refactorings than *mapping transformations* that transform into a different target model. For the realization and description of refactoring transformations, he uses his own language called *SMW* that operates on the UML metamodel — when the paper was written, there were no widely adopted transformation languages available. Dobrzański [36] provides a comprehensive survey on UML model refactorings in his master's thesis that deals with the refactoring of executable UML models [37]. He introduces an initial refactoring catalog for executable UML models. The refactorings are formalized with pre and post conditions in OCL. According to him, the main difference in refactoring executable models is that the update of the behavioral aspects of the models has to be taken into account.

More recent work on UML model refactoring and transformation is often based on the *Eclipse Modeling Framework* (EMF) representation of UML models. Biermann et al. [38] present work on an EMF model transformation framework that is based on graph transformations. They show how the rich theory of algebraic graph transformation can be applied to EMF model transformations. Using their method, the validation of the model transformations with respect to functional behavior and correctness is possible. They demonstrate their approach by using selected state chart refactorings. Similarly, Folli and Mens [39] suggest the use of graph transformations for model refactoring as well and present, as a proof-of-concept, how they have implemented a number of more complex UML model refactorings using the *AGG* [40] graph transformation tool.

6.2 Graphical Refactorings

Graphical refactorings are applied when it is hard to read and understand the graphical notation of a UML model: that is the corresponding diagrams containing partial views of the UML model. There are a huge variety of generic graph layout algorithms, and graph drawing itself is a very active research topic. Summaries can be found in a variety of textbooks, for example, Graph Drawing by Battista et al. [41]. Work on layouts of UML diagrams is rare. Ambler [30] provides informal guidelines that lack a systematic transformation mechanic to improve diagrams. However, it is arguable whether graphical refactorings should only change parts of a model using the refactoring mechanic or whether UML diagram specific transformations for complete optimal layouts are more desirable. Eichelberger and Gudenberg [28] discuss existing automatic layout methods for class diagrams and present their approach to laying out class diagrams that respect aesthetic rules, such as those described by [27]. Castello et al. [42] propose an automatic layout algorithm that improves the readability of state chart diagrams. It reduces the number of edge crossings and edge bends.

7 Tool Support

It is encouraged to use tools for measuring metrics, detecting smells, and applying refactorings to UML models. Manual application of refactorings, for example, is very error-prone and there is a risk that the changes are not semantically preserving due to human mistakes. Popular tools that support the automatic calculation of metrics and detection of bad smells in UML models are SDMetrics [43], Together [44], IBM Rational Systems Developer [45], and ArgoUML [46]. These tools partially use different terminologies for the term "bad smell". SDMetrics, for example, calls them *design rules*, Together calls them *audits*, or ArgoUML names them *design critics*. The toolset from Chaudron et al. [47] calculates metrics on UML models, it detects rules in sequence diagrams, it checks model consistency, and visualizes metrics in a metric viewer. Except for the commercial tool Poseidon for UML, which provides a refactoring browser supporting the refactorings from Boger at al. [48], none of the major commercial UML tools support refactoring beyond renaming and moving model elements. Tools that support more sophisticated UML refactorings are academic prototypes. An overview over existing academic UML refactoring tools is given by Dobrzański [36]. Van Gorp et al. [49] have implemented refactorings as plug-in for the Fujaba UML tool. Recently, several academic UML refactoring tools are evolving that build on EMF, for example, GaliciaUML [50].

8 Our Approach

The variety in the understanding and application of UML is visible in research as well. There is work dealing with UML diagrams only, while neglecting the fact that UML is a modeling language. On the other hand, others respect UML as a modeling language or mix descriptions based on the graphical notation with descriptions based on the UML metamodel. Most authors realize that there are relationships and dependencies between different diagrams that have to be respected when applying UML refactorings. To describe those relationships based on diagrams, however, is the wrong approach and we strongly believe that metrics, smell detection, and refactoring for UML should be described in a formalism that works on the UML metamodel. This ensures precision regarding the actual underlying UML model and regards complete models rather than just partial views. Authors such as [35] have realized this as well, but neither do they present a complete quality engineering approach for UML models including assessment and improvement that can be applied in every iteration of a development process, nor were there any widely spread model transformation languages available that could be applied for such uses. With our experience in the quality assurance of TTCN-3 [2,51], we aim to provide an integrated quality engineering approach for UML (Fig. 5) that consists of two main parts: quality assessment and quality improvement. For the assessment, a quality model is used and metrics quantify quality characteristics of this model. For the improvement, smell detection is used for locating possible issues and refactoring is used to improve the

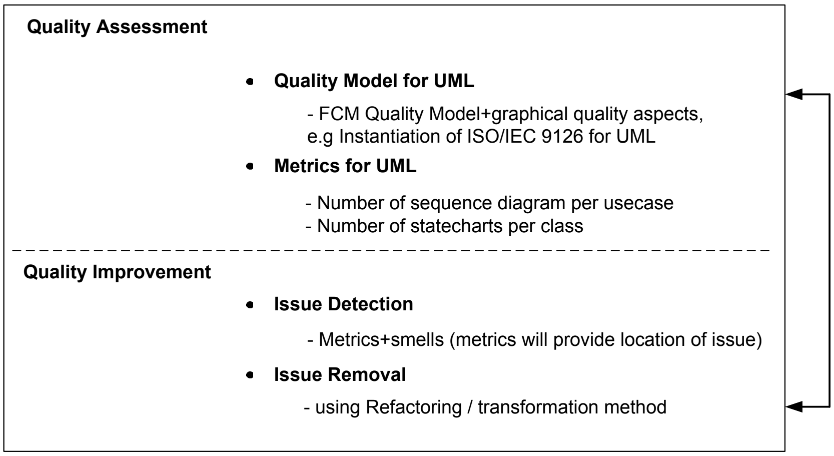

Quality Assessment
Quality Model for UML
- FCM Quality Model+graphical quality aspects, e.g Instantiation of ISO/IEC 9126 for UML
Metrics for UML
- Number of sequence diagram per usecase
- Number of statecharts per class
Quality Improvement
Issue Detection
- Metrics+smells (metrics will provide location of issue)
Issue Removal
- using Refactoring / transformation method

Fig. 5. Quality Assessment and Quality Improvement for UML

quality. Once the improvement step is completed, a quality reassessment quantifies whether the improvement was successful.

The first part in Fig. 5, i.e., the quality assessment requires a quality model for UML. We aim to use a generic FCM-based quality model, such as the ISO/IEC 9126 model, that can be instantiated with metrics to quantify its quality characteristics. The quality model has to be described in detail for its target environment of UML models, and (if necessary) it must be adapted for this domain. For example, the understandability characteristic of the ISO/IEC 9126 quality model does not only relate to the actual complexity of the UML model, but also to the graphical aspects of UML, such as the graphical smells described in Sect. 5.2. For the calculation of metrics, we use a widely used formalism such as OCL that works on the UML metamodel. While OCL is a language for expressing constraints in the first place, it can also be used to query models and to evaluate set sizes. For the quality improvement, we further plan to use languages that are now adopted for querying and transforming UML models. For the smell detection, we plan to provide a guideline catalog of bad smells in UML models using both an informal description in natural language and formal descriptions, e.g., given in OCL, where metrics with additional thresholds are defined to locate the smells. We are also evaluating the use of model transformation languages to identify smell locations by transforming the UML model into an instantiation of a metamodel that describes the results of the smell detection. For the actual refactorings, we plan to use existing model-to-model transformation languages such as *Query/View/Transformation* (QVT) [52], Xtend [53], or the *ATLAS Transformation Language* (ATL) [54] to describe the refactoring transformations, and OCL to define pre and post conditions for each refactoring. The techniques and languages described above are all based on the analysis at the model level instead of the graphical notation of UML. We plan to emphasize the actual model analysis instead of the graphical problems, as the involved layout techniques are part of a different field of research. However, graphical issues do play an important role in the overall assessment of the quality of a UML model — especially for the understandability quality characteristic. As we plan to use existing languages as formalisms to describe our metrics, smells, and refactorings, a proof-of-concept implementation will only involve the application of our metrics, smells, and refactoring descriptions to actual tool implementations that exist.

9 Status and Future Work

In our current work, we have successfully applied OCL for the calculation of metrics on a UML model and we have made first experiments to describe refactorings using Xtend. In both cases, we directly executed our experiments against existing UML models with tools that implement these languages. For the evaluation of OCL, we used the Eclipse OCL implementation of the *Eclipse Model Development Tools* project [55], which also allows the evaluation of the OCL result sets. For the refactoring, we have implemented so far simple refactorings using Xtend [53]. Our first experiments to detect smells and to apply refactoring using model-to-model transformation languages were successful, however, describing model-to-model transformations for UML models is not always an easy task due to the complexity of the UML metamodel. We plan to evaluate QVT and ATL to find out whether they make these descriptions more compact or more complicated and we also intend to evaluate the possibility of building a refactoring toolkit that eases the definition of refactorings. For the validation of our approach, we currently start to work on a case study that involves a UML model similar in size to industrial models. We then plan to apply our combined approach in an automated manner and use the quality reassessments after improvements to check whether the quality has improved. Expert reviews should then validate whether the automatic reassessment is correct.

References

1. Neukirchen, H., Zeiss, B., Grabowski, J.: An Approach to Quality Engineering of TTCN-3 Test Specifications. International Journal on Software Tools for Technology Transfer (STTT) 10(4), 309–326 (2008)
2. Neukirchen, H., Zeiss, B., Grabowski, J., Baker, P., Evans, D.: Quality Assurance for TTCN-3 Test Specifications. Software Testing, Verification and Reliability (STVR) 18(2) (2008)
3. Object Management Group (OMG): Meta Object Facility (MOF) Core Specification, Vers. 2.0, http://www.omg.org/docs/formal/06-01-01.pdf
4. Object Management Group (OMG): UML Infrastructure Specification, Vers. 2.2, http://www.omg.org/docs/formal/09-02-04.pdf
5. Object Management Group (OMG): UML Superstructure Specification, Vers. 2.2, http://www.omg.org/docs/formal/09-02-02.pdf
6. Abstratt Technologies: TextUML Toolkit, http://www.abstratt.com

7. Object Management Group (OMG): MOF 2.0/XMI Mapping, Vers. 2.1.1, http://www.omg.org/docs/formal/07-12-01.pdf
8. Object Management Group (OMG): OCL Core Specification Vers. 2.0, http://www.omg.org/docs/formal/06-05-01.pdf
9. Fenton, N., Pfleeger, S.: Software Metrics: A Rigorous and Practical Approach. PWS Publishing, Boston (1997)
10. McCall, J., Richards, P., Walters, G.: Factors in Software Quality. Technical Report RADC TR-77-369, US Rome Air Development Center (1977)
11. International Organization for Standardization (ISO) / International Electrotechnical Commission (IEC): ISO/IEC Standard No. 9126. Software Engineering- Product Quality; Part 1-4 (2001-2004)
12. Basili, V.R., Weiss, D.M.: A Methodology for Collecting Valid Software Engineering Data. IEEE Transactions on Software Engineering 10(6), 728–738 (1984)
13. Fowler, M.: Refactoring – Improving the Design of Existing Code. Addison-Wesley, Reading (1999)
14. van Emden, E., Moonen, L.: Java Quality Assurance by Detecting Code Smells. In: Proceedings 9th Working Conference on Reverse Engineering (WCRE 2002), pp. 97–106. IEEE, Los Alamitos (2002)
15. Parnas, D.: Software Aging. In: Proceedings of the 16th International Conference on Software Engineering (ICSE), pp. 279–287. IEEE Computer Society Press, Los Alamitos (1994)
16. Lange, C.: Assessing and Improving the Quality of Modeling. PhD thesis, Technische Universiteit Eindhoven, Netherland (2007)
17. Lange, C., Chaudron, R.: Managing Model Quality in UML-Based Software Development. In: Proceedings of the 13th IEEE International Workshop on Software Technology and Engineering in Practice (STEP 2005), pp. 7–16. IEEE Computer Society, Washington (2005)
18. Mohagheghi, P., Aagedal, J.: Evaluating Quality in Model-Driven Engineering. In: Proceedings of the International Workshop on Modeling in Software Engineering (MISE 2007), p. 6. IEEE Computer Society Press, Washington, DC (2007)
19. Pareto, L., Boquist, U.: A Quality Model for Design Documentation in Model-Centric Projects. In: Proceedings of the 3rd International Workshop on Software Quality Assurance (SOQUA 2006), pp. 30–37. ACM Press, New York (2006)
20. Lange, C.: Improving the Quality of UML Models in Practice. In: Proceedings of 28th International Conference on Software Engineering (ICSE 2006), pp. 993–996. ACM, New York (2006)
21. Chidamber, S.R., Kemerer, C.: A Metric Suite for Object-Oriented Design. IEEE Transactions on Software Engineering 20(6), 476–493 (1994)
22. Kim, H., Boldyreff, C.: Developing Software Metrics Applicable to UML Models. In: Proceedings of the 6th ECOOP Workshop on Quantitative Approaches in Object-Oriented Engineering (2002)
23. Baroni, A., Braz, S., Abreu, F.B.: Using OCL to Formalize Object-Oriented Design Metrics Definitions. In: Proceedings of the 6th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (2002)
24. McQuillan, J., Power, J.: A Metamodel for the Measurement of Object-Oriented Systems: An Analysis using Alloy. In: Proceedings of the 2008 international Conference on Software Testing, Verification (ICST), pp. 228–297. IEEE Computer Society Press, Washington (2008)
25. El-Wakil, M., El-Bastawisi, A., Riad, M.B., Fahmy, A.A.: A Novel Approach to Formalize Object-Oriented Design Metrics. In: Proceedings of the 9th International Conference on Empirical Assessment in Software Engineering (2005)

26. Kiewkanya, M., Muenchaisri, P.: Measuring Maintainability in Early Phase using Aesthetic Metrics. In: Dosch, W., Freund, R., Mastorakis, N. (eds.) SEPADS 2005, pp. 1–6. WSEAS, Stevens Point, Wisconsin (2005)
27. Purchase, H., Allder, J.-a., Carrington, D.: Graph Layout Aesthetics in UML Diagrams: User Preferences. Journal of Graph Algorithms and Applications 6(3), 255–279 (2002)
28. Eichelberger, H., von Gudenberg, J.W.: UML Class Diagrams - State of the Art in Layout Techniques. In: Proceedings of the International Workshop on Visualizing Software for Understanding and Analysis, Amsterdam (2003)
29. Gronback, R.: Model Validation: Applying Audits and Metrics to UML Models (2004), http://conferences.codegear.com/jp/article/32089
30. Ambler, S.: The Elements of UML 2.0 Style. Cambridge University Press, Cambridge (2005)
31. Astels, D.: Refactoring with UML. In: Proceedings of the 3rd International Conference on eXtreme Programming and Flexible Processes in Software Engineering, XP 2002 (2002)
32. Fowler, M.: Refactoring: Improving the Design of Existing Code. Addison-Wesley, Reading (1999)
33. France, R., Bieman, J.: Multi-View Software Evolution — A UML-based Framework for Evolving Object-Oriented Software. In: Proceedings of 17th IEEE International Conference on Software Maintenance (ICSM 2001). IEEE Computer Society Press, Los Alamitos (2001)
34. Sunyé, G., Pollet, D., Traon, Y., Jézéquel, J.: Refactoring UML Models. In: Gogolla, M., Kobryn, C. (eds.) UML 2001. LNCS, vol. 2185, pp. 134–148. Springer, Heidelberg (2001)
35. Porres, I.: Model Refactorings as Rule-Based Update Transformations. In: Stevens, P., Whittle, J., Booch, G. (eds.) UML 2003. LNCS, vol. 2863, pp. 159–174. Springer, Heidelberg (2003)
36. Dobrzański, L.: UML Model Refactoring- Support for Maintenance of Executable UML Models. Master's thesis, Blekinge Institute of Technology, School of Engineering, Ronneby, Sweden (2005)
37. Mellor, S., Balcer, M.: Executable UML: A Foundation for Model-Driven Architecture. Addison-Wesley, Reading (2002)
38. Biermann, E., Ermel, C., Taentzer, G.: Precise Semantics of EMF Model Transformations by Graph Transformation. In: Czarnecki, K., Ober, I., Bruel, J.-M., Uhl, A., Völter, M. (eds.) MODELS 2008. LNCS, vol. 5301, pp. 53–67. Springer, Heidelberg (2008)
39. Folli, A., Mens, T.: Refactoring of UML models using AGG. In: Proceedings of the 3rd International ERCIM Symposium on Software Evolution (2007)
40. Taentzer, G.: A Graph Transformation Environment for Modeling and Validation of Software. In: Pfaltz, J.L., Nagl, M., Böhlen, B. (eds.) AGTIVE 2003. LNCS, vol. 3062, pp. 446–453. Springer, Heidelberg (2004)
41. Battista, G.D., Eades, P., Tamassia, R., Tollis, I.G.: Graph Drawing - Algorithms for the Visualization of Graphs. Prentice-Hall, Englewood Cliffs (1998)
42. Castello, R., Mili, R., Tollis, I.: Automatic Layout of Statecharts. Software Practice & Experience 32(1), 25–55 (2002)
43. SDMetrics: The Software Design Metrics tool for the UML, http://www.sdmetrics.com
44. Borland: Borland Together, http://www.borland.com/us/products/together
45. IBM: IBM Rational Systems Developer, http://www.ibm.com/software/awdtools/developer/systemsdeveloper

46. ArgoUML Project: ArgoUML, http://argouml.tigris.org
47. Lange, C., Chaudron, R.: Empanada: Empirical analysis of architecture and design quality, http://www.win.tue.nl/empanada/tools.htm
48. Boger, M., Sturm, T., Fragemann, P.: Refactoring Browser for UML. In: Aksit, M., Mezini, M., Unland, R. (eds.) NODe 2002. LNCS, vol. 2591, pp. 366–377. Springer, Heidelberg (2003)
49. Gorp, P., Stenten, H., Mens, T., Demeyer, S.: Towards Automating Source-Consistent UML Refactorings. In: Stevens, P., Whittle, J., Booch, G. (eds.) UML 2003. LNCS, vol. 2863, pp. 144–158. Springer, Heidelberg (2003)
50. Seuring, P.: Design and Implementation of a UML Model Refactoring Tool. Master's thesis, Hasso-Plattner-Institute for Software Systems Engineering at the University of Potsdam (2005)
51. Zeiss, B., Vega, D., Schieferdecker, I., Neukirchen, H., Grabowski, J.: Applying the ISO 9126 Quality Model to Test Specifications – Exemplified for TTCN-3 Test Specifications. In: Proceedings of Software Engineering 2007 (SE 2007). Lecture Notes in Informatics (LNI), vol. 105, Köllen Verlag (2007)
52. Object Management Group (OMG): Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, http://www.omg.org/docs/formal/08-04-03.pdf
53. openArchitectureWare.org: openArchitectureWare (oAW), http://www.openarchitectureware.org
54. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I., Valduriez, P.: ATL: A QVT-Like Transformation Language. In: Companion to the 21st ACM SIGPLAN Symposium on Object-Oriented Programming Systems, Languages, and Applications. ACM Press, New York (2006)
55. Eclipse Foundation: Eclipse Model Development Tools (MDT) OCL, http://www.eclipse.org/modeling/mdt/?project=ocl

Optimal String Edit Distance Based Test Suite Reduction for SDL Specifications

Gábor Kovács[1], Gábor Árpád Németh[1], Mahadevan Subramaniam[2], and Zoltán Pap[3]

[1] Department of Telecommunications and Media Informatics – ETIK, Budapest University of Technology and Economics, Magyar tudósok körútja 2, H-1117, Budapest, Hungary
{kovacsg,gabor.nemeth}@tmit.bme.hu

[2] Computer Science Department, University of Nebraska at Omaha Omaha, NE 68182, USA
msubramaniam@mail.unomaha.edu

[3] Ericsson Telecomm. Hungary, H-1117 Budapest, Irinyi J. u. 4-20, Hungary
zoltan.pap@ericsson.com

Abstract. We propose a test selection method that provides efficient test sets for systems based on SDL specifications. Our approach builds on previous results of Voung et al. and Feijs et al. on string edit distance based coverage metrics. The method reduces a set of test cases represented in the MSC (Message Sequence Chart) notation, while maintaining the highest possible distance between all pairs of traces defined by the given test set. The algorithm is tunable by a parameter representing the threshold distance for test redundancy. We show that the algorithm runs in polynomial time of the size of the input test set and that it is independent of the size of the system. We implemented and incorporated the algorithm into our SDL-based test selection framework, and evaluated against existing symbol coverage and fault coverage based test selection approaches by conducting experiments on the well-known INRES and Conference Protocol. Results indicate that the string edit distance based method yields similar results in terms of reduction-capability and coverage as the other approaches, but with significantly less complexity.

Keywords: SDL based test selection, string edit distance, MSC test cases.

1 Introduction

From the design perspective, system development facilitated by formal modelling has several benefits, mainly due to the higher level of abstraction it provides to system architects. A further and often overlooked advantage of modeling techniques comes from the testing phase of the development life cycle: A correct model of the system under development may serve as a basis for automating test development, which is a crucial and increasingly expensive part of the software development process.

Research in the field of testing has established various test generation methods [1,2,3] and tools like Autolink [4], TorX [5], TGV [6] or Phact [7] for different formalisms including extended finite state machine (EFSM) based modelling languages such as SDL (Specification and Description Langugage). These methods typically have a sound mathematical background, but are often criticized for resulting in excessive number of test cases under industrial-size application. A major challenge of automatic test generation is, therefore, the detection and reduction of redundancies among the large number of test cases derived from a system model. This type of problem is addressed by test selection methods [2,8,9,10,11]: their objective is to minimize the cardinality of the target test set without sacrificing its quality. However as shown in [8] the selection problem is NP-hard, so approximative solutions must be defined [10,11].

In this paper we present a method for automatic test selection building on a promising idea first published by Voung et al. [9] and later improved by Feijs et al. [12]. Vuong's original approach defines string edit distance measures and addresses the normalization of traces represented as strings to approximate differences among patterns of system behavior; traces are considered to be similar if their distance is smaller than a given parameter. The latter paper generalizes that original idea by introducing a cycling and a reduction heuristics and gives formulae to precisely calculate the distance between traces containing finite number of traversals of loops around a state in the specification.

We utilize the string edit distance-based test coverage metric to reduce MSC (Message Sequence Chart) test sets. A two-step selection algorithm is proposed to find the minimum cardinality subset with the highest possible diversity: First the minimum cardinality for a given approximation threshold is determined by reducing the distance based selection problem to an assignment problem in bipartite graphs. Then a test set with the highest overall internal edit distance is selected from all subsets with that computed cardinality. We show that the algorithms run in polynomial time of the size of the input test set and that it is independent of the size of the system. Furthermore, the paper proposes an iterative test generation method that can be used in incremental development of compact test sets.

The algorithm has been implemented and customized to accept the SDL specification of the system under test and a test set defined in MSC. The solution has been incorporated into our SDL-based test selection framework [2], and has been evaluated against existing symbol coverage and fault coverage based test selection approaches by conducting experiments on the well-known INRES [13] and Conference Protocol [7]. Results indicate that the string edit distance based method does not show significant differences in terms of reduction-capability and coverage compared to the other approaches, but requires significantly less computation time.

The rest of the paper is organized as follows. A brief overview of our assumptions and notations is given in Sect. 2. Section 3 describes the procedure of distance maximization among the test cases. In Section 4 we introduce how the proposed method can be used to reduce the size of the resulting test set during an automatic test generation process. We demonstrate our method through an example. Section 5 compares the string edit distance based test selection method with two other appoaches: a fault based and a coverage based solution using the sample SDL systems INRES and Conference Protocol. Section 6 presents our conclusions.

2 Preliminaries

SDL [14] is a well-accepted world standard supported by the ITU (International Telecommunication Union). SDL is widely used in the telecommunications industry for the description of telecommunication protocols, but it may be used in other fields where high reliability is required. Typically complex, event-driven, real-time and communicating systems can be effectively described in SDL.

SDL is built on the extended finite state machine (EFSM) model that is an extension of the basic FSM formalism by adding support for the use of variables. An EFSM is a 5-tuple: $EFSM = (S, I, O, V, H)$, where S is the finite set of states, I is the finite set of input symbols, O is the finite set of output symbols, V is the finite set of variables, and H is the finite set of transitions. Each transition $h \in H$ is a 6-tuple: $h = (s_j, i, P(V), A(V), o, s_k)$, where $s_j \in S$ is the start state of the transition, $i \in I$ is an input, P is a set of predicates on the variables, A is a set of actions on the variables, $o \in O$ is an output and $s' \in S$ is the next state.

2.1 Test Cases and Their String Representation

Existing test generation tools like Autolink [4] provide excellent means to derive large number of MSC traces from SDL specifications. In practice the number of such executable traces is infinite and therefore exhaustive testing based on all traces is generally impossible. The purpose of test selection (and test generation in its essence) is to identify a subset of trace set sufficient to establish a required level of confidence in the correctness of the system.

Throughout this paper we consider SDL specifications with a reliable reset signal as a test start point. A test set consists of a set of MSCs, where each test case is a finite trace after a reset input. The MSC traces in the examples are generated by means of random-walk, but any other test derivation tool or algorithm may be considered as well. Note that in this paper we use the terms test sets and trace sets interchangeably.

Traces derived from an SDL specification can be represented as strings on an arbitrary alphabet C. A mapping $M : \{I \cup O\}^+ \rightarrow C$ defines a set of pairs of an SDL signal sequence and a character of alphabet C: $\langle (x_{i1} x_{i2} \ldots x_{im}), c_i \rangle$, where $x_{ik} \in I \cup O, k = 1 \ldots m, k \in \mathbb{N}, c_i \in C$. Note that according to this definition several successive input or output signals may be mapped to a character of that given alphabet. Such mapping M is the marked trace notation defined in [12].

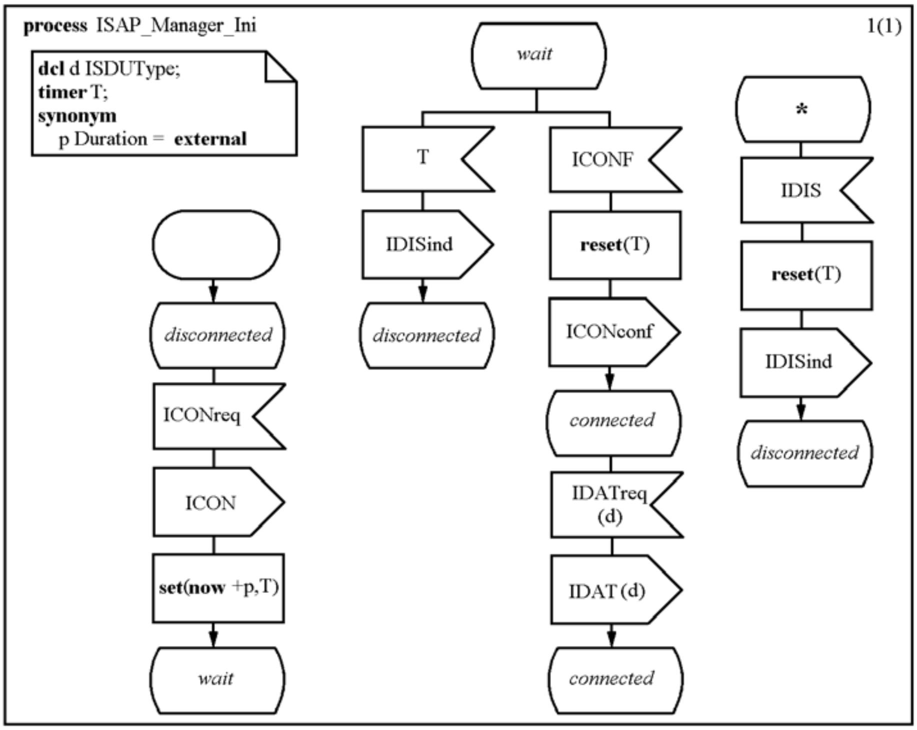


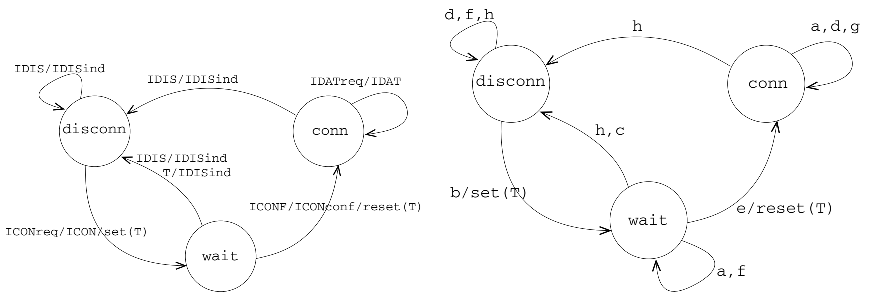
Fig. 1. INRES ISAP Manager Ini process (see [13])

Let us use the `ISAP Manager Ini` process of the INRES system [13] as example, its SDL representation can be seen in Fig. 1. The process has three states $S = \{$`disconnected`, `wait`, `connected`$\}$, the initial state is `disconnected`. The machine is not completely specified, input signals such as the `IDATreq` in state `disconnected` are not shown and considered to be implicitly consumed, that is, they do not change the state of the machine and do not produce output. The input and output signal sets are the following: $I = \{$`ICONreq`, `T`, `ICONF`, `IDATreq`, `IDIS`$\}$ and $O = \{$`ICON`, `IDISind`, `ICONconf`, `IDAT`$\}$. The timeout `T` is considered to be an input signal and according to the specification it is only possible in state `WAIT`.

Let us assume a simple signal sequence to character mapping M defined by Table 1, which maps each possible transition of the process `ISAP Manager Ini` to a character. The table indicates implicit events as well: dash represents that no output signal is produced. The state transition diagram of the unfolded FSM and the graph with mappings applied are presented in Fig. 2(b). Note that the cycling heuristics of [12] is not used in the mapping for the sake of simplicity. EFSM aspects can be taken into account when defining the mapping for parameterized signals: a parameterized signal name can be mapped to a set of different characters depending on the actual parameter values.

Table 1. Signal sequence to character mapping for ISAP Manager Ini

`ICONreq/−`	a
`ICONreq/ICON`	b
`T/IDISind`	c
`ICONF/−`	d
`ICONF/ICONconf`	e
`IDATreq/−`	f
`IDATreq/IDAT`	g
`IDIS/IDISind`	h



(a) INRES `ISAP Manager Ini` process (b) INRES `ISAP Manager Ini` process after event to character mapping

Fig. 2. The state transition diagram of the FSM of the ISAP Manager Ini process from the sample INRES protocol in original form and after event to character mapping

Figure 2(a) shows the state transition graph of the `ISAP Manager Ini` when that is unfolded into a finite state machine, and Fig. 2(b) shows the graph after applying the event to string mapping defined above.

2.2 Trace Distance

The edit distance between strings σ_1 and σ_2 is the minimum number of edit operations (character insert, character delete and character overwrite) needed to transform σ_1 to σ_2 [15]. Different edit operations may have distinct costs assigned, for the sake of simplicity we consider unit edit operator costs in the current paper.

Let Σ be a finite set of strings over the alphabet C, and let $d : \Sigma \times \Sigma \rightarrow \mathbb{R}$ be a distance metric such that $d(\sigma_1, \sigma_1) = 0$, $d(\sigma_1, \sigma_2) = d(\sigma_2, \sigma_1)$ and $d(\sigma_1, \sigma_3) \leq d(\sigma_1, \sigma_2) + d(\sigma_2, \sigma_3)$, for all $\sigma_1, \sigma_2, \sigma_3 \in S$. If the lengths of σ_1 and σ_2 are l_1 and l_2 respectively, the time and space complexity of computing the distance is $O(l_1 l_2)$. The distance metric computation above is according to one defined in [15], but other approaches can be considered as well.

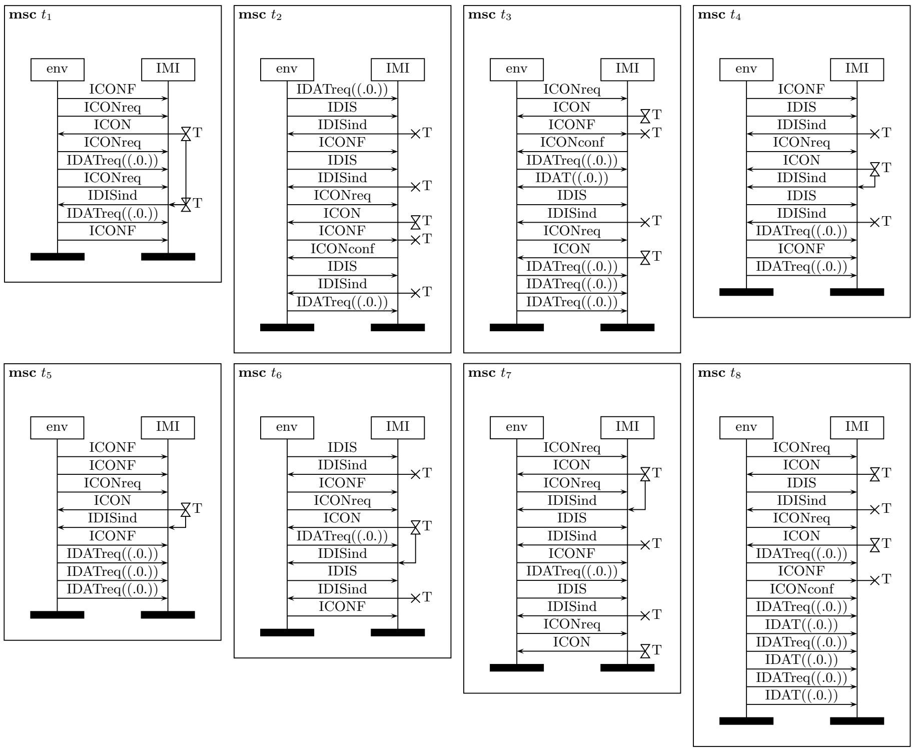


Fig. 3. The T test set of eight MSCs generated with random walk. (IMI abbreviates `ISAP Manager Ini` from the INRES system)

Let $\mathbf{D} = [d_{ij}]$ be a distance matrix of the set of MSC traces T. Let $d_{ij} = d(M(t_i),\ M(t_j))$, where $1 \leq i, j \leq |T|, i, j \in \mathbb{N}$. Note that $\mathbf{D}$ is symmetric because of the symmetric nature of distance and the size of the matrix is $|T| \times |T|$. Let $\mathbf{D}_U$ and $\mathbf{D}_L$ denote the strictly upper and lower triangular matrices of $\mathbf{D}$ respectively.

The metric space involves the normalization of edit distance values as it is done in [9] and [12]. For the sake of simplicity – and without any loss of generality – in the example of this paper we dispense with the normalization by using traces of a given length and consider that the distances are non-negative integer numbers not greater than that length.

Example. Let the test set T be composed of the MSCs of Fig. 3 derived with random walk for the `ISAP Manager Ini` process. According to the M mapping of Table 1 $M(T) = \{dbafacfd,\ fhdhbehf,\ beghbfff,\ dhbchfdf,\ ddbcdfff,$ $hdbafchd,\ bachdfhb,\ bhbfeggg\}$ are the string representations of T. Note that reset inputs are omitted in the figure for the sake of clarity, and all test sequences have reset input as prefix thus start from initial state `disconnected`. These eight strings together traverse 11 of the 13 transitions of the machine in Fig. 2(b).

The distance matrix of this trace set $M(T)$ with unit cost edit operations is:

$$\mathbf{D} = \begin{bmatrix} 0&8&7&6&6&3&6&7\\ 8&0&5&5&7&6&6&7\\ 7&5&0&6&5&8&5&6\\ 6&5&6&0&3&6&6&6\\ 6&7&5&3&0&6&6&7\\ 3&6&8&6&6&0&6&7\\ 6&6&5&6&6&6&0&7\\ 7&7&6&6&7&7&7&0 \end{bmatrix}$$

3 String Edit Distance Based Test Suite Reduction

This section proposes a method for selecting a test set with the highest possible diversity with regard to the distance based coverage metric. The inputs of the method are the $\mathbf{D}$ distance matrix of the traces, and an ε approximation parameter. The method consists of two stages: First the minimum cardinality of the target test set is calculated assuming the ε threshold, then the test set with the maximum internal distance is selected.

For our discussions we assume the notion of ε-approximation as defined by Feijs et. al. in [12]: T' is an ε-cover of T, where

$$T' \subseteq T, \varepsilon \geq 0 \iff \forall t \in T : \exists t' \in T' : d(M(t), M(t')) \leq \varepsilon.$$

This implies that for each T and ε there exists at least one minimal cardinality T' ε-cover of T, for which $\forall t'_i, t'_j \in T' : d(M(t'_i), M(t'_j)) > \varepsilon, t'_i \neq t'_j$. The test set T is divided into two disjoint subsets: a T'' subset of test cases that can and a $T'_0 \subseteq T'$ subset of test cases that can not be ε-covered by other test cases. The test cases from T'_0 must be included in T', and from T'' the minimum number of cases must be selected: $T' = T'_0 \cup \mathsf{reduce}(T'')$. Thus the maximum reduction of T'' yields the most compact T'.

Algorithm 1 finds the size of the most compact test suite T' that can be achieved with the string edit distance based selection in polynomial time for a given distance matrix. The inputs of the algorithm are the distance matrix of T and a ε threshold parameter. For the computation two matrices A and A_μ of boolean values and a bipartite graph $G' = (N'_R \cup N'_C, E')$ are used. The algorithm first determines which test cases of T ε-cover each other, and if they do, it is marked in $\mathbf{A}$ with 1 value (lines 4-7). Then, the size of the T'_0 set is determined in lines 8-10. In lines 11-15, if exactly the same coverage is found for two test cases, then one of them is eliminated. Lines 16-22 construct a bipartite graph G' based on matrix $\mathbf{A}$ such that the set of rows of $\mathbf{A}$ is mapped to the node set N'_R and the set of columns of $\mathbf{A}$ is mapped to the node set N'_C, and if a_{ij} is not 0, then there is an edge from n'_i to n'_j, where $n'_i \in N'_R, n'_j \in N'_C$. The Hopcroft-Karp algorithm [16] is used for finding a maximum cardinality assignment μ in G' (line 23). The k minimal size returned in line 28 is the size

Algorithm 1. Reducing the string edit distance based test case selection problem to a matching problem in bipartite graphs

```
input  : $\mathbf{D}$ distance matrix of $T$ test set; $\varepsilon$ threshold
output: $k$, the maximum number of redundant cases for the given $\varepsilon$
1  data( $\mathbf{A} = [a_{ij}], a_{ij} \in \{0,1\}; \mathbf{A}_\mu = [a_{\mu ij}], a_{\mu ij} \in \{0,1\}$;
2  $G' = (N', E')$ graph, where $N' = N'_R \cup N'_C$, $\forall n_i, n_j \in N'_R : (n_i, n_j) \notin E'$,
   $\forall n_i, n_j \in N'_C : (n_i, n_j) \notin E'$)
   /* Initialization */
3  $k := 0, N'_R := \emptyset, N'_C := \emptyset, E' := \emptyset, \mathbf{A} := 0, \mathbf{A}_\mu := 0$;
   /* Computing the $\mathbf{A}$ matrix */
4  foreach $i, j, 1 \le i,j \le |T|$ do
5      if $d_{ij} < \varepsilon$ then $a_{ij} = 1$;
6      else $a_{ij} = 0$;
7  endfch
   /* Counting the elements of $T'_0$ */
8  foreach $i$ do
9      if $\sum_j a_{ij} = 0$ then $k := k + 1$;
10 endfch
   /* Finding test cases with the same coverage */
11 foreach $k, l, 1 \le k \le |T| - 1, k < l \le |T|$ do
12     if $\sum_j (a_{kj} \mathsf{xor} a_{lj}) = 0$ then
13         foreach $j, 1 \le j \le |T|$ do $a_{lj} := 0, a_{jl} := 0$;
14     endif
15 endfch
   /* Constructing the $G'$ graph */
16 forall the $t_i \in T$ do
17     $N'_R := N'_R \cup t_i$;
18     $N'_C := N'_C \cup t_i$;
19 endfall
20 foreach $i, j, 1 \le i,j \le |T|$ do
21     if $a_{ij} > 0$ then $E' := E' \cup \{(n'_i, n'_j)\}, n'_i \in N'_R, n'_j \in N'_C$;
22 endfch
23 Let $E'_\mu$ be the matching selected by the Hopcroft-Karp algorithm;
   /* Marking the pairs of the maximum matching in $\mathbf{A}_\mu$ */
24 foreach $i,j, 1 \le i,j \le |T|$ do
25     if $(n'_i, n'_j) \in E'_\mu$ then $\mathbf{A}_{\mu\,\mathsf{ij}} = \mathbf{1}$;
26     else $\mathbf{A}_{\mu\,\mathsf{ij}} = \mathbf{0}$;
27 endfch
28 return $k := k + \mathsf{rank}(A^L_\mu)$
```

of T'_0 plus the rank of the upper or lower triangual matrix of $\mathbf{A}_\mu$ constructed based on the maximum matching in lines 24-27.

The complexity of the construction of $\mathbf{A}$ (lines 4-7) and G' (lines 16-22) are $O(|T|^2)$. The worst-case complexity of the Hopcroft-Karp algorithm [16] applied to the graph $G' = (N', E')$ in line 23 is $O(\sqrt{|N'|}|E'|)$. Since $|N'| \le 2|T|$ and

$$\mathbf{A}_{\varepsilon=\mathbf{5}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & \boxed{1} & 0 & 0 \\ 0 & 0 & 1 & \boxed{1} & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & \boxed{1} & 0 \\ 0 & \boxed{1} & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ \boxed{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \boxed{1} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

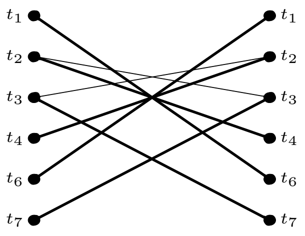


Fig. 4. The **A** matrix and the bipartite graph G' constructed from it

$|E'| \leq |T|^2$ according to lines 16-22, its complexity is $O(|T|^{5/2})$. Hence the worst case complexity of this algorithm is determined by the search for same rows in **A** in lines 11-15, which is $O(T^3)$.

Example. The ε-coverage of matrix **D** of the example at the end of Sect. 2.2 with $\varepsilon = 5$ (note that we dispense with normalization) and the bipartite graph G' constructed from **A** are in Fig. 4. In matrix **A** on the left the eighth is the only row that contains only 0 values, therefore $T_0' = \{t_8\}$. The second and the fifth row are identical, so we may remove either t_2 or t_5 from the further processing as a redundant case. The bipartite graph G' and the matching problem constructed from T'' without the redundant t_5 can be seen on the right. Edges show that a trace ε-covers an other test case. Bold lines select a maximum cardinality matching: the selected elements of $\mathbf{A}_\mu$ boxed in the matrix **A** are $(1,6)$,$(2,4)$,$(3,7)$,$(4,2)$,$(6,1)$,$(7,3)$. The rank of the upper or lower triangular matrices of $\mathbf{A}_\mu$ is 3, therefore beside t_5, three additional test cases can be removed: $T_U'' = \{t_1, t_2, t_3, t_5\}$ or $T_L'' = \{t_4, t_5, t_6, t_7\}$. The resulting T' candidates are $T_1' = \{t_1, t_2, t_3, t_8\}$, if t_5 was considered to be redundant, or $T_2' = \{t_1, t_3, t_5, t_8\}$, if t_2 was considered to be redundant and $T_3' = \{t_4, t_6, t_7, t_8\}$. In general more maximum cardinality assignments may exist, but all with the same cardinality. □

According to [9] and [12] two patterns of behavior are approximated to be less similar if the distance between their string representation is greater. The redundancy among the test cases in a test set is the least, if the sum of all pairwise distances is maximal. Hence, if more than one minimal cardinality T' solutions exist, the one with the maximum internal distance should be preferred.

Selecting the T' test set with minimal cardinality from the test set T, such that T' is an ε-cover of T and the test cases differ from each other as much as possible can be calculated in a polynomial time of $|T|$. This means that the sum of distances between all pairs of the test cases of T' is maximal, therefore the optimization problem is:

$$\max \sum_{t_i \in T', t_j \in T'} d(M(t_i), M(t_j)), \qquad (1)$$

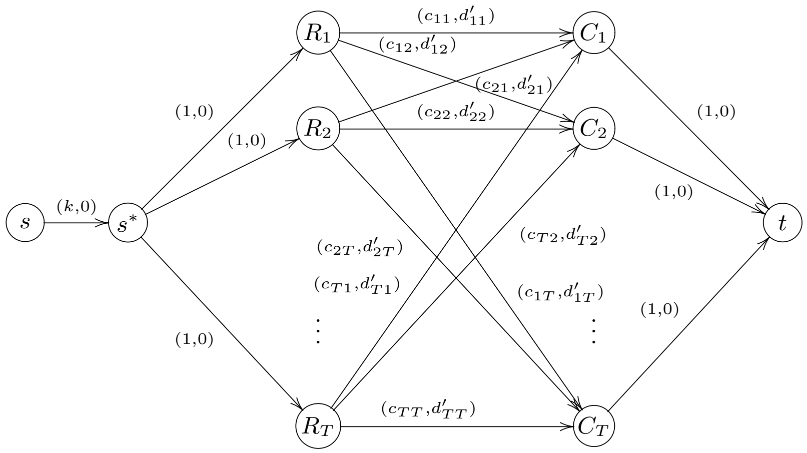


Fig. 5. The flow problem equivalent to the maximum distance k-cardinality matching

where $\forall i, j : d(M(t_i), M(t_j)) > \varepsilon$. A T' with the minimum k cardinality can be determined with Algorithm 1. There may be many T's with this k cardinality, and there is at least one with the maximum distance sum.

This problem is equivalent to the following minimum cost maximum flow problem. Let $\mathbf{C} = [c_{ij}]$, where $c_{ij} \in \{0, 1\}$ a capacity matrix, and let the distance matrix $\mathbf{D} = [d_{ij}]$ of T be a cost matrix. Let $G' = (N', E')$ be a bipartite graph of distance matrix $\mathbf{D}$ as constructed in Algorithm 1. Let $G = (N, E)$ be directed weighted graph extending G' such that $N = \{s, s^*\} \cup N' \cup \{t\}$ and $E = \{(s, s^*), (s^*, n'_i)\} \cup E' \cup \{(n'_j, t)\}$, for all $n'_i \in N'_R, n'_j \in N'_C$ and let all edges between nodes N'_R and N'_C be directed from n'_R to n'_C. Each edge is assigned with a capacity value and a cost defined as follows. Let the capacity of all edges $e \in E$ be $c = 1$, except for (s, s^*) that has a capacity of k. Hence, the maximum flow capacity is determined by the $\{(s, s^*)\}$ cut and it equals to k. Let the cost of edges (s, s^*),(s^*, n'_i) and (n'_j, t) be 0, and let the cost of edges (n'_i, n'_j), where $n'_i \in N'_R, n'_j \in N'_C$, be $d'_{ij} = \max_{i,j}(d_{ij}) - d_{ij}$ for all i and j. See Fig. 5.

The minimum cost maximum flow can be found by solving the following optimization with linear programming, and can be expressed as

$$\max \textstyle\sum_{i=1}^{|T|} \sum_{j=1}^{|T|} c_{ij} d_{ij}$$

$$\text{where} \quad \sum_{j=1}^{|T|} c_{ij} \leq 1, i = 1, ..., |T|, \qquad \sum_{i=1}^{|T|} c_{ij} \leq 1, j = 1, ..., |T|,$$
$$\sum_{i=1}^{|T|} \sum_{j=1}^{|T|} c_{ij} = k, \text{ and} \qquad c_{ij} \in \{0, 1\}.$$

This problem has been defined as the k-cardinality assignment problem by Dell'Amico and Martello and has been shown to be a P-space problem in [17].

Example. It has been shown that $k = 4$ for the $\varepsilon = 5$ case and therefore the "best" solution contains $|T| - k = 8 - 4 = 4$ test cases. In the example three solutions have been found. The sum of distances between pairs of T_1' is 40, for T_2' and T_3' it is 38, so T_1' is the best reduced suite.

4 Application to Selective Automatic Test Generation

This section proposes a selective extension to automatic test generation methods. When a new test case is derived from an SDL specification, before adding that to the test set as usual, the method proposed in Sect. 3 is used to find redundancies immediately. Note that this method may lead to suboptimal solution compared to the single stage optimization, but the resulting test set is already reduced and makes test selection unnecessary.

We generalize the distance function to evaluate the effect of merging a new trace into an already existing trace set. Let $d : (t', T) \rightarrow \mathbb{R}$ compute the distance between the trace t' and the trace set T such that $d(t', T) = \min_i d(t', t_i)$, where $t_i \in T$.

Algorithm 2. Selective automatic test generation

```
input  : $\varepsilon$; $K$ iteration limit; $Sp$ specification
output: $T$ set
1 $T[0] := \emptyset$;
2 $k := 1$;
3 repeat
4     $t' :=$ derive($Sp$);
5     if $d(t', T) < \varepsilon$ then $T[k] :=$ reduce($T[k-1] \cup t'$);
6     else $T[k] := T[k-1] \cup t'$ $k := k + 1$;
7 until $k > K$ ;
8 return $T[k]$
```

Algorithm 2 is a selective method for automatic test generation. The inputs of the method are a ε selection threshold, a K iteration limit, which is an upper bound for the number of generated test cases. The output is a reduced test set. The algorithm can co-work with any test generation algorithm [4,5,6,7]. In each iteration cycle a new test case t' is derived. That test case is added to the test set immediately if its distance from every element of the test set is greater than the given threshold. Otherwise the union of the newly generated test case and the old test set is optimized with the method of Sect. 3 to maintain the density of the test set.

Example. In the example let the iteration limit be $K = 8$, and let MSC traces from Fig. 3 be generated iteration by iteration. In general the iteration limit, the length and the number of sequences are independent, but setting all these configuration parameters to 8 simplifies this example.

Two cases are investigated, the $\varepsilon = 5$ and the $\varepsilon = 6$ case. The string edit operations insert, delete and replace are assumed to have unit cost. The iterative test generation procedure is presented only for the first case. The distance matrix for the test cases of this example can be found in Sect. 2.2, and the coverage matrix can be seen Fig. 4.

Let us first investigate the $\varepsilon = 5$ case. This $\mathbf{A}_{\varepsilon=\mathbf{5}}$ matrix implies that the resulting set consists of four strings.

Step 1. The first string is $t' = dbafacfd$, so $T_{\varepsilon=5}[1] = \{dbafacfd\}$. Since this is the only element of T in step 1, $\mathbf{D}_{\varepsilon=5}[1] = 0$ and $\mathbf{A}_{\varepsilon=5}[1] = 0$. This sequence traverses 6 of the 13 transitions of the FSM in Fig. 2(b).

Step 2. In this step the string $t' = fhdhbehf$ is added to $T_{\varepsilon=5}[1]$, thus $T_{\varepsilon=5}[2] = \{dbafacfd, fhdhbehf\}$ and

$$\mathbf{D}_{\varepsilon=5}[2] = \begin{bmatrix} 0 & 8 \\ 8 & 0 \end{bmatrix} \quad \mathbf{A}_{\varepsilon=5}[2] = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

The second sequence traverses also 6 transitions and two sequences together traverse 9 different transitions.

Step 3. The new string added to T is $t' = beghbfff$. This new string traverses 5 different transitions and provides one new, not yet traversed transition for $T_{\varepsilon=5}[2]$. However $d(t_2[2], t') = 5$, that is, these two sequences ε-cover each other, therefore one of them is dropped despite of providing a new transition.

$$\mathbf{D}_{\varepsilon=5}[3] = \begin{bmatrix} 0 & 8 & 7 \\ 8 & 0 & 5 \\ 7 & 5 & 0 \end{bmatrix} \quad \mathbf{A}_{\varepsilon=5}[3] = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & \boxed{1} & 0 \end{bmatrix}$$

As the $d(t_1[2], t_2[2]) = 8 > d(t_1[2], t') = 7$, $t_1[2]$ and $t_2[2]$ are kept.

Step 4. The situation is the same as in step 3. The new string $t' = dhbchfdf$ traverses 5 transitions from which none is new for $T_{\varepsilon=5}[3]$.

$$\mathbf{D}_{\varepsilon=5}[4] = \begin{bmatrix} 0 & 8 & 6 \\ 8 & 0 & 5 \\ 6 & 5 & 0 \end{bmatrix} \quad \mathbf{A}_{\varepsilon=5}[4] = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & \boxed{1} & 0 \end{bmatrix}$$

As the $d(t_1[3], t_2[3]) = 8 > d(t_1[3], t') = 6$, $t_1[3]$ and $t_2[3]$ are kept.

Step 5. In this step $t' = ddbcdfff$ is added to the sequence set, that contains $T_{\varepsilon=5}[5] = \{dbafacfd, fhdhbehf, ddbcdfff\}$. This sequence traverses only 4 different transitions which are already traversed by $T_{\varepsilon=5}[4] = T_{\varepsilon=5}[2]$, thus a redundant sequence is added to the set.

Step 6. The sixth string to be added is $t' = hdbafchd$. However $d(t_1[5], t') = 3$, therefore one of them (t_1 or t') is considered to be redundant:

$$\mathbf{D}_{\varepsilon=5}[6] = \begin{bmatrix} 0 & 8 & 6 & 3 \\ 8 & 0 & 7 & 6 \\ 6 & 7 & 0 & 6 \\ 3 & 6 & 6 & 0 \end{bmatrix} \quad \mathbf{A}_{\varepsilon=5}[6] = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \boxed{1} & 0 & 0 & 0 \end{bmatrix}$$

Because $d(t_1[5], t_2[5]) + d(t_1[5], t_3[5]) = 14 > d(t', t_2[5]) + d(t', t_3[5]) = 12$, $t_1[5]$ is kept and t' is dropped.

Step 7. Though the sequence $t' = bachdfhb$ traversing 6 different transitions is completely redundant for $T_{\varepsilon=5}[6]$, it is added to the set, because $\mathbf{A}_{\varepsilon=5}[7] = 0$. This t' also does not increase the number of traversed transitions.

Step 8. The last sequence $t' = bhbfeggg$ is also added to $T_{\varepsilon=5}[8]$ which is now $T_{\varepsilon=5}[8] = \{dbafacfd, fhdhbehf, ddbcdfff, bachdfhb, bhbfeggg\}$. This last sequence traverses two more transitions so $T_{\varepsilon=5}[8]$ traverses the 11 of the 13. This resulting set contains five sequences, one more as it would have been necessary according to $\mathbf{A}_{\varepsilon=5}$. The final distance matrix is:

$$\mathbf{D}_{\varepsilon=5}[8] = \begin{bmatrix} 0 & 8 & 6 & 6 & 7 \\ 8 & 0 & 7 & 6 & 7 \\ 6 & 7 & 0 & 6 & 7 \\ 6 & 6 & 6 & 0 & 7 \\ 7 & 7 & 7 & 7 & 0 \end{bmatrix}$$

When setting now $\varepsilon = 6$ instead of 5, the resulting set is further reduced. According to the $\mathbf{A}_{\varepsilon=6}[8]$ below two more traces can be removed: $t_3[8]$ and $t_4[8]$. (Note that in the $\varepsilon = 5$ case these two cases were named previously as redundant.) The remaining three traces $T_{\varepsilon=6}[8] = \{dbafacfd, fhdhbehf, bhbfeggg\}$ still traverse 11 of the 13 transitions.

$$\mathbf{A}_{\varepsilon=6}[8] = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & \boxed{1} & 0 & 1 & 0 \\ \boxed{1} & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

When using $\varepsilon = 6$ from the beginning of the iteration cycle, the final trace set is $T_{\varepsilon=6} = \{dbafacfd, fhdhbehf, bhbfeggg\}$. Without the iterative cycle the resulting set would be $T_{\varepsilon=6} = \{dbafacfd, fhdhbehf\}$ with 9 transitions traversed.

The transition coverage is the same for both the $\varepsilon = 5$ and the $\varepsilon = 6$ cases. In step 3 one new transition is discovered, but the trace is dropped due to redundancy. Two new transitions are found in step 8 including the one dropped in step 3.

The total number of distance calculations is 20 in the $\varepsilon = 5$ case and 13 in the $\varepsilon = 6$ case. Without the iterative cycle it would be 28, hence matrix $\mathbf{A}$ is provided with less calculations. The gain depends on the relation between trace lengths and the total number of traces generated.

5 Empirical Analysis

This section gives simulation based evaluation of the – single-staged – string edit distance based test case selection method. A process from each of two systems, the `ISAP Manager Ini` process from the INRES [13] and the Conference

Table 2. Properties of the unfolded SDL systems and input parameters for the test selection methods

Name	States	Inputs	Outputs	Non-implicit transitions	Faults injected	Symbols used in mapping
INRES	3	5	4	7	25	8
Conference	5	11	3	19	78	22

Protocol [7] both represented in SDL, and test sets derived manually and automatically are used in this investigation. The result of the method proposed in this section are compared to the results of two different selection strategies based on pairing of test cases and test case requirements [8]. One of these two methods is the fault coverage based test selection method proposed in [2], where the test case requirements are considered to be faults injected systematically into the system according to the given fault model. The other is a transition coverage based method that regards the checking of a transition of an FSM as a test case requirement. Hence the two SDL systems are unfolded into FSMs.

Table 2 characterizes the two unfolded SDL systems. The INRES `ISAP Manager Ini` process' FSM consists of three states, has four inputs, a timer and four outputs. The total number of transitions is 15, the number of different faults that can be injected to that process according to the fault model proposed in [2] is 25. The input and output events are mapped to eight characters. The Conference Protocol limited to three users and at most one conference at a time has five states, eleven inputs and three outputs when unfolded into an FSM. 16 of the 55 transitions are non-implicit, and the same fault model yields 78 mutant systems. 22 characters are used in the mapping to represent the input and output events of this system.

Three test sets are used to evaluate the selection method proposed in this section. One is an automatically generated set of the eight cases that has already been used in the example in Sect. 2.1. The second is an automatically generated one of thirteen cases. These first two sets are executed against the `ISAP Manager Ini` process. A third automatically generated set is run againsts the Conference Protocol.

The results of simulations can be seen in Table 3. In the experiment on the INRES system's `ISAP Manager Ini` process with eight automatically generated test cases, the string edit distance based method and the transition coverage based method provide exactly the same three test cases. These three cases miss four faults that could be detected with the fault coverage method. As automatically generated cases are more likely to check implicit transitions their transition coverage is greater than in case of the manual cases. In the experiment with 13 manually generated cases, the string edit distance based method achieves a bigger reduction (13 to 6 vs. 13 to 8) than the transition coverage based method at the cost of not traversing all transitions. Both provide the same fault coverage, but miss four injected faults that could be found.

Table 3. Results of the selection experiments

Test set	System	Method	Number of selected cases	Number of faults detected	Transitions covered
Automatic (8)	INRES	String	4/8	19/25	11/13
		Transition	3/8	19/25	11/13
		Fault	4/8	23/25	11/13
Manual (13)	INRES	String	6/13	19/25	6/13
		Transition	8/13	19/25	10/13
		Fault	4/13	23/25	8/13
Automatic (40)	Conference	String	14/40	59/78	39/55
		Transition	11/40	59/78	40/55
		Fault	6/40	60/78	25/55

In case of the Conference Protocol all three methods provide nearly the same fault coverage. The biggest reduction is achieved by the fault coverage based method, but that traverses the least transitions.

In general according to the experiments shown above, the smallest reduced test and the highest fault coverage ration is provided by the fault based method. The highest transition coverage can be achieved by the transition coverage based method. The string based method provides results close to the transition coverage based method, but this is automatic and requires much less algorithmic steps to execute than the automatic fault based method.

6 Conclusions

This paper proposes an efficient approach for both single stage and iterative test selection processes. A method is provided to select a subset of a test set in polynomial time by searching for similar patterns of events. The approach builds on previous results of string edit distance based test selection methodology. We propose two selection criteria: one to identify the minimum cardinality of the target test set for a given a ε-cover, and an other to select the test cases that differ from each other as much as possible assuming the string distance based metric. An iterative procedure is given to reduce the computation requirement for long test cases, which is specially suited for maintaining test sets for regression testing. The operation of the algorithm is demonstrated through an example.

The method is compared with fault and coverage based test selection techniques using the sample systems INRES and Conference Protocol. The string edit distance based method provides similar fault detection capability as the transition coverage based selection, and requires less computation than the other two methods.

A generalization of the method is possible for test cases represented as a labeled, rooted, unordered trees. By calculating the distance matrix of such trees the method is applicable for test case selection without fundamental changes.

References

1. Lee, D., Yiannakakis, M.: Principles and methods of testing finite state machines – a survey. Proceedings of the IEEE 43(3), 1090–1123 (1996)
2. Kovács, G., Pap, Z., Le, V.D., Wu-Hen-Chang, A., Csopaki, G.: Applying mutation analysis to SDL specifications. In: Reed, R., Reed, J. (eds.) SDL 2003. LNCS, vol. 2708, pp. 269–284. Springer, Heidelberg (2003)
3. Wong, W.E., Restrepo, A., Qi, Y., Choi, B.: An EFSM-based test generation for validation of SDL specifications. In: AST 2008: Proceedings of the 3rd international workshop on Automation of software test, pp. 25–32. ACM, New York (2008)
4. Schmitt, M., Grabowski, J., Hogrefe, D., Koch, B.: Autolink – a tool for the automatic and semi-automatic test generation. In: Wolisz, A., Schieferdecker, I., Rennoch, A. (eds.) Formale Beschreibungstechniken für verteilte Systeme, Nr. 315, GMD-Studien. GMD-Forschungszentrum Informationstechnik GmbH (1997)
5. Tretmans, G., Brinksma, H.: Torx: Automated model-based testing. In: Hartman, A., Dussa-Ziegler, K. (eds.) First European Conference on Model-Driven Software Engineering, pp. 31–43 (2003)
6. Jard, C., Jeron, T.: TGV: Theory, principles and algorithms. In: 6th World Conference on Integrated Design and Process Technology, IDPT 2002 (2002)
7. Heerink, L., Feenstra, J., Tretmans, J.: Formal test automation: The conference protocol with phact. In: Ural, H., Probert, R.L., von Bochmann, G. (eds.) 13th IFIP International Conference on Testing of Communicating Systems (TestCom 2000), pp. 211–220. Kluwer Academic, Dordrecht (2000)
8. Harrold, M., Gupta, R., Soffa, M.: A methodology for controlling the size of a test suite. Transactions on Software Engineering and Methodology 2(3), 270–285 (1993)
9. Vuong, S.T., Alilovic-Curgus, J.: On test coverage metrics for communication protocols. In: von Bochmann, G., Dssouli, R., Das, A. (eds.) Proceedings of the IFIP TC6/WG6.1 Fourth International Workshop on Protocol Test Systems IV, pp. 31–45. North Holland, Amsterdam (1992)
10. Csöndes, T., Kotnyek, B., Szabó, J.: Application of heuristic methods for conformance test selection. European Journal of Operational Research 142(1), 203–218 (2001)
11. Williams, A., Probert, R.: Formulation of the interaction test coverage problem as an integer program. In: Schieferdecker, I., König, H., Wolisz, A. (eds.) Proceedings of the IFIP 14th international Conference on Testing Communicating Systems XIV. IFIP Conference Proceedings, vol. 210, pp. 283–298. Kluwer, B.V., Deventer (2002)
12. Feijs, L., Goga, N., Mauw, S., Tretmans, J.: Test selection, trace distance and heuristics. In: Schieferdecker, I., König, H., Wolisz, A. (eds.) Proceedings of the IFIP 14th international Conference on Testing Communicating Systems XIV. IFIP Conference Proceedings, vol. 210, pp. 267–282. Kluwer, B.V., Deventer (2002)
13. Ellsberger, J., Hogrefe, D., Sarma, A.: SDL Formal Object-oriented Language for Communicating Systems. Prentice-Hall, Englewood Cliffs (1997)
14. International Telecommunications Union: Recommendation Z.100 (11/07), Specification and Description Language (SDL), http://www.itu.int/rec/T-REC-Z.100/en
15. Wagner, R., Fischer, M.: The string-to-string correction problem. Journal of the ACM 21(1), 168–173 (1974)
16. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press and McGraw-Hill (2001)
17. Dell'Amico, M., Martello, S.: The k-cardinality assignment problem. Discrete Applied Mathematics 76(1), 103–121 (1997)

Early Schedulability Analysis with Timed Use Case Maps

Jameleddine Hassine

Cisco Systems, 3000 Innovation Drive, Kanata, Ontario, K2K 3J9, Canada
jhassine@cisco.com

Abstract. The ability to perform quantitative analysis at the requirements level supports the detection of design errors during the early stages of a software development life cycle. This would help reduce the cost of later redesign activities in case of unsatisfactory performance. This paper presents a novel approach to perform schedulability analysis at the requirement stage using Timed Use Case Maps (TUCM) language. The proposed approach relies on the computation of Worst-Case Execution Time (WCET), resource allocation and scheduling policies. Timing and resource constraints are first incorporated into UCM specifications, then mapped to Abstract State Machines (ASM) formalism and implemented in AsmL language, allowing for simulation and schedulability analysis. The applicability of the approach is illustrated using an example of the Automatic Protection Switching (APS) feature.

1 Introduction

The ability to perform quantitative analysis at the requirements level supports the detection of design errors during the early stages of a software development life cycle. Thus, reducing the cost of later redesign activities. In order to achieve this goal, non-functional aspects dealing with reliability, availability, performance, and timing constraints have to be incorporated at the software requirement phase. This is essential in order to correctly model, for instance, time dependent applications at early stages in system development. Typical classes of such applications are communication protocols and real-time distributed systems, which have to satisfy stringent timing constraints. Missing such constraints (such as the failure to meet a deadline) may impact system reliability and therefore can have serious consequences. Hence, appropriate methods are needed to complement the software specification approaches, so that requirement specifications can be analyzed in terms of schedulability. Schedulability analysis methods are usually used to validate the fulfillment of time constraints of an application (deadline, response time, ready time, etc.).

The widespread interest in time modeling and analysis techniques, provides the major motivation for this paper. There is particular focus on the need for a formal approach to validate time-based scenarios during the requirement phase. This paper serves different purposes:

– It provides an Abstract State Machine (ASM) [1] based formal semantics for the Timed Use Case Maps (TUCM) language [2] based on a discrete time model. The resulting timed ASM models are implemented using AsmL language allowing for analysis, simulation and validation.

- It provides a novel approach to perform schedulability analysis based on the Timed Use Case Maps language. The proposed approach relies on the computation of Worst-Case Execution Time (WCET), resource allocation and scheduling policies.
- It extends the ongoing research towards the construction of a formal framework for the Use Case Maps language [3] to describe, simulate and analyze real-time systems [2,4,5].

In an attempt to make this paper self-contained, a brief introduction to schedulability analysis is provided in Sect. 2. Section 3 outlines the UCM time extensions introduced in Hassine et al. [2]. Section 4 represents the core of the paper, that presents the timed UCM based schedulability approach, discusses UCM resource modeling (Sect. 4.1), explains how the worst case execution time (WCET) is computed (Sect. 4.3), presents the adopted scheduling strategy (Sect. 4.4), and provides the corresponding ASM-based semantics (Sects. 4.2 and 4.5). An example of the Automatic Protection Switching (APS) feature is presented in Sect. 5. Section 6 describes some related work. Finally, limitations and future work are discussed in Sect. 7 followed by conclusions in Sect. 8.

2 Schedulability Analysis

Schedulability analysis addresses the problem of meeting the specified timing requirements in order to have an understandable, predictable and maintainable system timing behavior. It represents a powerful tool for checking the correctness of real-time systems at design time. By definition real-time system deadlines must not be missed. To ensure this property, schedulability analysis allows checking timing constraints given the resources (hardware and software) available for the execution of the system tasks.

A system is said to be schedulable, under a given scheduling strategy, if it can be guaranteed never to miss a deadline. Ramamritham and Stankovic [6] have identified four main classes of real-time scheduling strategies: off-line vs. on-line, preemptive vs. non-preemptive, static vs. dynamic and optimal vs. heuristic scheduling. Some of the most popular scheduling strategies include:

- *Priority-driven Preemptive Scheduling*: Liu and Layland [7] proposed two scheduling algorithms. The first algorithm is a fixed priority algorithm, called *Rate Monotonic*. It assigns static priorities to tasks based on their periods (i.e. higher priorities are given to tasks with shorter periods). The dispatcher will make sure that at any time, the highest priority task is actually running. Liu and Layland have also analyzed a dynamic scheduling strategy, called *Earliest Deadline First (EDF)*, where the highest priority is assigned to the task with the closest absolute deadline.
- *Dynamic Planning-based Scheduling* [8]. Unlike the priority-driven approaches, feasibility is checked at run-time where a task is accepted for execution only if it is guaranteed to meet its time constraints. A task is *guaranteed* by constructing a plan for task execution given a set of assumptions such as the task worst case execution time, resource requirements, timing constraints, and the nature of faults in the system.

- *Value-Based Scheduling* (VBS) [9]: VBS is a more advanced scheduling approach where each task is associated with a reward and penalty that is offered to the system depending on whether the task meets or misses its deadline. It represents a means by which the system copes with degradation gracefully (i.e. in case of overloaded system for instance) by executing critical tasks that offer high values/benefits/rewards to the functioning of the system.

Most of the schedulability analysis methods rely on knowing for each task of the system the upper bound of its execution time, commonly called *Worst-Case Execution Time (WCET)*. There are two main classes of methods for WCET analysis [10]: static and measurement-based methods.

1. Static methods: These methods consist of analyzing a system statically and do not rely on real execution or on simulation. They are based on static program analysis [11] of possible control flow paths. Static methods emphasize *safety* by producing guaranteed bounds on the execution time [10].
2. Measurement-based methods: These methods consist of executing the tasks on the targeted hardware or on a simulator for a subset of the inputs. The measurements of a subset of all possible executions do not actually produce bounds, but provide estimates of the bounds from maximal and minimal observed execution times.

In what follows, a brief overview is provided of the Timed Use Case Maps language, initially introduced in [2], which represents the base for implementing the proposed schedulability analysis approach.

3 The Timed Use Case Maps Language

The Use Case Maps language (UCM) is a high-level scenario based modeling technique that can be used to capture and integrate functional requirements in terms of causal scenarios representing behavioral aspects at a high level of abstraction. UCM can also provide stakeholders with guidance and reasoning about a system-wide architecture and behavior. This is being reflected by Use Case Maps being part of the ITU-T standard User Requirements Notation (URN) Z.151 [3].

The Use Case Maps language [3] has been extended with time constraints [2]. Two formalization approaches[1] for the Timed Use Case Maps language have been proposed:

1. Clocked Transition System (CTS) based semantics: Based on a discrete time model, CTS [12] provides an easy and flexible way to reason about system execution over time. Hassine et al. [2] defined two step semantics (with two sets of transition rules) for timed UCM models aiming to cover both interleaving and true concurrency models. However, no executable model was provided.
2. Timed Automata (TA) based semantics: Based on a dense time model, the authors in [5] defined a timed automaton [13] template for each timed UCM construct. Furthermore, timed UCM models expressed in timed automata can be validated and verified using the UPPAAL model checker [14]. Although, an executable model was provided allowing for formal verification of timed UCM models, the concept of resources was overlooked and no schedulability based verification was considered.

[1] For a detailed description of timed UCM formalization, see [2,5,15].

To accommodate the proposed schedulability approach, a slightly modified version of the time criteria introduced in [2] is used. The following summarizes the selected time-related criteria:

- *Initiation and termination of enabling* [16] represents a flexible and suitable choice for UCM level of abstraction. Both a lower and upper bound may be imposed on the enabling of a responsibility. A responsibility may be enabled any time between *minDL* and *maxDL* time units after the completion of its predecessor. Such a delay is introduced in order to describe, for instance, situations of queueing delay or when the resources needed to execute a responsibility are not immediately available.
- *Durational semantic model* where time is mainly consumed by responsibilities. A responsibility may be associated with *minDur* and *maxDur* denoting respectively its best and worst case execution times. UCM control constructs such as OR-Forks (involving condition evaluation) may take some time to complete.
- *A global and centralized clock* for measuring and increasing time globally over the system is used: MasterClock (*MClock*).
- *A discrete time model* is adopted. The smallest time unit (i.e. clock tick) used to track system evolution over time is named δ. It defines the granularity of Master-Clock (*MClock*).
- *Both* relative *and* absolute *time models* are considered. Relative time is used to define the duration of responsibilities and their incurred delay. Absolute time is used to track the value of the MasterClock (*MClock*). It is used in start points to record the scenario starting time and to define responsibilities' deadlines.

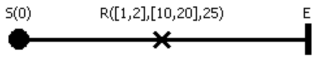


Fig. 1. Delayable responsibility

- *Urgency* such that a responsibility is considered as *urgent* when enabled immediately after the execution of its predecessor (i.e. *minDL*= *maxDL* = 0). Alternatively, it is considered as *delayable*. All UCM control constructs (such as OR-Fork, OR-Join, AND-Fork, etc.) are considered as urgent once enabled. Transitions are processed as soon as they are enabled allowing for a maximal progress. Figure 1 illustrates a delayable responsibility *R* with an estimated delay between 1 and 2, an execution interval between 10 and 20 and a deadline of 25 (when *MClock = 25*). The start point *S* is triggered at *MClock = 0.*

4 Timed UCM-Based Schedulability Approach

Figure 2 illustrates how schedulability analysis fit into the proposed UCM-based requirement specification and validation framework. The UCM functional model, the timed annotations (discussed in Sect. 3) and the binding architecture (for the UCM components) are formalized using Abstract State Machines [1] and implemented using AsmL [17], an advanced ASM-based executable specification language developed by the Foundation of Software Engineering Group at Microsoft Research.

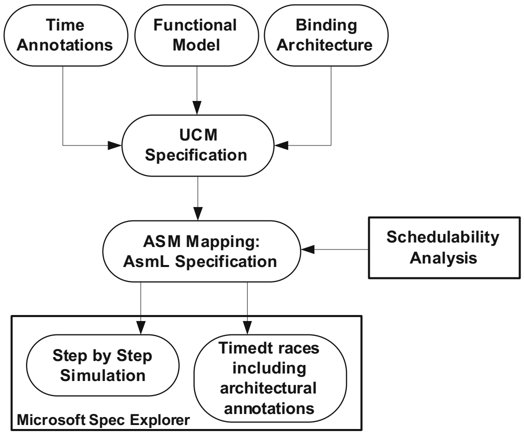


Fig. 2. Schedulability analysis in the UCM-based requirement specification flow

4.1 Resource Modeling

In UCM, components are generic and can represent software entities (objects, processes, etc.) as well as non-software entities (e.g. processors). Each component can handle a set of resources (power, memory, communication bandwidth, etc.).

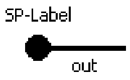
The URN standard [3] distinguishes between processing resources (that execute or process operations) and passive resources (that do not perform operations). A processing resource can be a processor, a disk, or a digital signal processor (DSP). In the proposed schedulability analysis context, only generic processing resources are considered. In addition, URN does not distinguish best and worst execution times and misses the notions of delays and priorities, which are needed for schedulability analysis. The approach in this paper assumes the following:

– A resource is defined as a global quantity that has a finite size (such as 1 Gigabit of memory).
– How much of a given resource a UCM construct consumes can be specified as an absolute interval [R_{min}, R_{max}] (e.g. [128,256] kilobytes of memory) or as a percentage interval [P_{min},P_{max}] (e.g. [5%,10%] of CPU utilization).
– Resources are consumed by UCM constructs (responsibilities and control constructs) during scenarios execution. For example, if a responsibility consumes 20 kb of memory, the total memory usage will be increased by 20 kb during the responsibility execution and will be decreased by 20 kb when completed.

4.2 ASM-Based Formal Semantics for Timed UCMs

The proposed semantics are based on the author's previous work [4] on untimed ASM based semantics for the Use Case Maps language. The following presents the timed ASM rules for start points, responsibilities and OR-fork constructs.

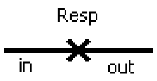
- **Start points.** If the *PreCondition* is satisfied, there occurs at least one event from the *triggeringEvent-set* and no additional delay is required (that is $minDL \leq MClock \leq maxDL$), then the start point is triggered and the control passes to the outgoing edge *out*. Figure 3 describes the start point ASM rule.



```
if CurrentConstruct is StartPoint(PreCondition, Trigger, SP-Label, in, out,
minDL, maxDL, Component) then
  if (PreCondition & EvaluateTrigger & (minDL $\leq$MClock $\leq$ maxDL) then
    add out to active
    remove in from active
where:
EvaluateTrigger: TriggerringEvent-set $\times$ {events} $\rightarrow$ Boolean;
evaluates whether the set of events occurring at StartPoint are included in the
TriggeringEvent-set.
```

Fig. 3. ASM rule for a timed start point

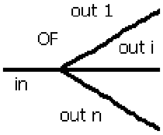
- **Responsibilities.** If the control is on the incoming edge (*in*), the master clock is increased by the value of the actual delay ($random(minDL, maxDL)$). Subsequently the master clock is increased by the value of the actual execution time ($random(minDur, maxDur)$) and the resource utilization is updated. Upon completion, the component resources are released and the control is transferred to the outgoing edge (*out*). Figure 4 illustrates the responsibility ASM rule. The resource utilization (*ResUtilization*) and the remaining resources (*remainingRes*) are abstract and can be refined in terms of CPU or memory utilization.



```
if CurrentConstruct is Responsibility (in, Resp, out, minDL, maxDL, minDur,
maxDur, minU, maxU, deadline, Comp) then
    DelayTime:= random (minDL,maxDL)
  step
    MClock := MClock + DelayTime
    Resp
    ExecutionTime:= random (minDur,maxDur)
    ResUtilization:= random (minU, maxU)
  step
    MClock := MClock + ExecutionTime
    Comp.remainingRes := Comp.remainingRes - ResUtilization
  step
    if (MClock $\geq$ delay) then Write('Deadline Misssed')
    if (Comp.remainingRes $\leq$ 0) then Write('Not enough resources')
    Comp.remainingRes := Comp.remainingRes + ResUtilization
    add out to active
    remove in from active
```

Fig. 4. ASM rule of a timed responsibility

– **OR-Fork.** If the control is on the incoming edge of an OR-Fork, the conditions are evaluated and the control passes to the edge associated with the true condition. If more than one condition evaluates to true (nondeterministic choice), the control passes randomly to one of the outgoing edges associated to the true conditions. At all times a set of active edges (*active*) is maintained. The set *active* is updated by adding the out_i that corresponds to the true condition and by removing the edge *in*. The necessary resources are allocated then freed after the outgoing edge selection is completed. Figure 5 illustrates the OR-Fork ASM rule.



```
if CurrentConstruct is OR-Fork(in, $[Cond_i]_{i\leq n}$,$[out_i]_{i\leq n}$, minDur, maxDur,
minU, maxU, Comp)
then
    MClock := MClock + random(minDur, maxDur)
    ResUtilization:= random (minU, maxU)
  step
    Comp.remainingRes := Comp.remainingRes - ResUtilization
  step
   if (Comp.remainingRes $\leq$ 0) then Write('Not enough resources')
    Comp.remainingRes := Comp.remainingRes + ResUtilization
    add (choose k in $[out_k]_{k\leq l}$) to active
    remove in from active
```

Fig. 5. ASM rule of a timed OR-fork

4.3 Worst Case Execution Time Computation

In the ASM rules presented in the previous section, a random delay and a random execution time are applied by the function *random*. Similarly, an average delay and execution time can be applied. Although, it gives a fair execution time estimate, the average execution time is not the main factor when optimizing an application. If a system allocates enough execution time to execute its worst case execution path under all circumstances, anything using less time than that makes the system idle for the rest of the allocated time. This makes it important to minimize the WCET.

For a given timed UCM scenario, where initial values and triggered start points are provided, the worst case execution time is simply the summation of the individual construct execution times and delays along the traversed path. Figure 6 illustrates an example of a timed UCM with dependent control flows, where responsibilities R1, R2, R3

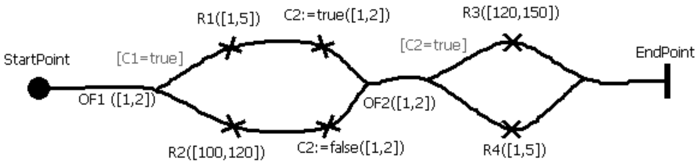


Fig. 6. UCM with dependent control flows

and R4 have a WCET of 5, 120, 150, 5 respectively, condition evaluation at OR-Forks OF1 and OF2 have a WCET of 2 each and variable assignments have a WCET of 2 each. All constructs are considered as urgent (i.e. delay=0). The simple calculation of WCET is:

$$\begin{aligned} WCET &= OF1 + max(R1 + (C2 := true), \\ &\quad R2 + (C2 := false)) + OF2 + max(R3, R4) \\ &= 2 + max(5 + 2, 120 + 2) + 2 + max(150, 5) \\ &= 276 \end{aligned}$$

However, if a static analysis of the control flow is performed, the WCET will be:

$$\begin{aligned} WCET &= max(OF1 + R1 + (C2 := true) + OF2 + R3, \\ &\quad OF1 + R2 + (C2 := false) + OF2 + R4) \\ &= max(2 + 5 + 2 + 2 + 150, 2 + 120 + 2 + 2 + 5) \\ &= 161 \end{aligned}$$

The computed WCET is a less pessimistic, but still accurate WCET, as it is still an upper bound for the actual execution time. Since the proposed approach is based on run-time simulation, the computed WCET represents a fair upper bound value to the scenario execution time.

Deriving a WCET estimate for every single UCM construct depends on the resources binding model. Indeed, WCET of a responsibility varies from one computing platform to another. Multiple potential component allocations can be considered when deriving WCET for a UCM construct.

4.4 Scheduling Strategy

When a UCM construct is ready to be executed on a sequential path, the required memory and CPU are allocated, the construct is executed, then the WCET is increased by the maximum delay and maximum duration values (i.e. WCET := WCET + maxDL + maxDur). Then resources are released and the control passes to the subsequent construct. In the presence of concurrent paths (see Fig. 7) and contrary to the strategy of the untimed version [4], where the next edge to be executed is chosen randomly from the set of active edges, in this work the selection of the subsequent edge (leading to the next UCM construct) is based on the following criteria (in the order they appear below):

1. The responsibility with the highest priority is selected.
2. In the case of responsibilities having the same priority, the edge leading to the construct with the minimum delay is selected.
3. In the case of equal priority and delay, the construct having the minimal duration is selected.

The ASM-based scheduler implements a priority-driven non-preemptive scheduling with interleaving semantics. Hence, once a construct is selected for execution, it will run to completion allowing for maximal progress. However, before executing a responsibility, a check (deadline $\geq$ MClock + maxDur) whether it meets its deadline is performed. If there is a risk that it will miss its deadline, the user is warned by including

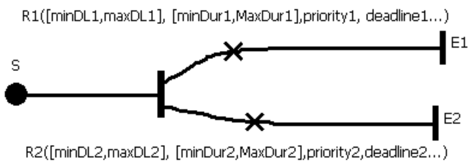

R1([minDL1,maxDL1], [minDur1,MaxDur1],priority1, deadline1...)
S
E1
E2
R2([minDL2,maxDL2], [minDur2,MaxDur2],priority2,deadline2...)

Fig. 7. A UCM concurrent path

a warning message in the execution trace, but the responsibility can still be executed. Similarly, if a responsibility requires more resources than the system can offer, the user is warned.

Based on the selected scheduling strategy, the schedulability of a timed end-to-end UCM scenario is defined as follows:

Definition 1 (Schedulability). *A timed UCM scenario is said to be schedulable if and only if it satisfies the following conditions:*

- *The UCM scenario reaches its end points.*
- *All responsibilities can meet their deadlines.*
- *System resources are not exhausted.*
- *Its end-to-end WCET is acceptable.*

After ensuring that a system is schedulable, a new responsibility will not be admitted unless it passes a schedulability test.

4.5 AsmL Implementation of Timed UCM Semantics

Data Structures. The data structures introduced in [4] are extended to cover time and resources annotations. Figure 8 describes an excerpt of the AsmL implementation

```
structure UCMConstruct
    case SP_Construct
        in_hy as HyperEdge
        out_hy as HyperEdge
        label as String
        preCondition as BooleanExp
        minDL as Integer
        maxDL as Integer
        location as Component

    case R_Construct
        in_hy as HyperEdge
        out_hy as HyperEdge
        label as String
        minDL as Integer
        maxDL as Integer
        minDur as Integer
        maxDur as Integer
        minU as Integer
        maxU as Integer
        deadline as Integer
        priority as Integer
        location as Component
```

```
    case OF_Construct
        in_hy as HyperEdge
        Selec as Set of OR_Selection
        label as String
        minDL as Integer
        maxDL as Integer
        location as Component
    ...
structure Component
    Name as Integer
    Type as String
    MemorySize as Integer
    RemainingMem as Integer
    RemainingCPU as Integer

structure OR_Selection
    out_hy as HyperEdge
    out_cond as BooleanExp

structure UCMElement
    source as UCMConstruct
    hyper as HyperEdge
    target as UCMConstruct
```

Fig. 8. Excerpt of the AsmL implementation of the timed UCM data structures

of *UCMConstruct* data structure that incorporates many case statements as a way of organizing different variants of UCM constructs. *UCMElement* specifies how UCM constructs are connected to each other – represented as (source, edge, target). A UCM component has (1) static properties such as its *Name*, *Type* (Processor, DSP, etc.) and *MemorySize*, and (2) dynamic properties (updated during scenario execution) such as the remaining memory (*RemainingMem*) and the available CPU (*RemainingCPU*).

AsmL Implementation of the Scheduler. Assuming a single-agent based solution with interleaving semantics, Figure 9 presents an excerpt of the AsmL implementation of the scheduler.

```
class Agent
    const id as String
    var active as Edge
    var mode as Mode
Program()
step
   until ((act = {}) or (me.mode = inactive))
   do
    let h = {t1.edge || t1 in act }
    let z = GetNext(h) choose s2 in z.level.ele where HyperExists(z.edge, GetInHyperEdge(s2.source))
match (s2.source)       choose z in act where z.delay= minimumDL
     choose h in level.ele where HyperExists(active, GetInEdge(h.source))
      match (s2.source)
// Rule of Start Point
      SP_Construct (a,b,c,d,e,f,g): step
           if d.Value() = true and (MClock ≤ f) and (MClock ≥ e)
            WriteLine(Start Point:+c+ in Component: + g)
            add activ(b, z.level) to act
            choose r in act where r.edge = a
            remove r from act
// Rule of Responsibility
       R_Construct (a,b,c,d,e,f,g,l,i,k,l,m): step
            MClock := MClock + random(d,e)
            Utilization := random(l,i)
           step
            if (m.RemainingU ≤ Utilization)
              WriteLine('Responsibility ' + a + 'cannot be scheduled: Not enough CPU ')
            ExecuteResponsibility((s2.source) as R_Construct)
            m.RemainingU := m.RemainingU - Utilization
           step
            MClock := MClock + random(f,g)
            WCET := WCET + e + g
           step
            if (WCET ≥ k)
              WriteLine('Responsibility ' + a + ': missed its deadline k')
           add activ(b, z.level) to act
           choose r in act where r.edge = a
           remove r from act
            m.RemainingU := m.RemainingU + Utilization
// ...
```

Fig. 9. Excerpt of the AsmL implementation of the scheduler

4.6 Properties of the Proposed Schedulability Approach

Interpretation vs. Compilation. The proposed timed ASM-UCM scheduler is based on the interpretation concept of execution. The scheduler looks at each construct of the specification, works out what it means, executes its corresponding rule, notifies the user of any violations, and then proceeds with the next construct. With interpretation the modification of the semantics of a timed UCM construct simply results in changing the corresponding ASM rule (without modifying the original specification). In a compilation-based execution, such a modification requires a redesign of the mapping of UCM constructs to the target formalism (such as LOTOS formal language) and the regeneration of the specification (executable form).

Scalability. The scheduler does not maintain system states during the execution of consecutive UCM constructs (except for the stub-plugin hierarchy). This would minimize the amount of state information managed by the scheduler allowing for better scalability.

Language Evolution. The proposed ASM rules can be easily modified to accommodate UCM language evolution. Indeed, the modification of the semantics of a UCM construct or the addition of a new construct result in the modification or the addition of a new ASM rule that describes the semantics of the new construct.

4.7 Simulation and Inspection of Traces

The use of traces or execution histories as an aid to debugging, testing and analysis is a well established technique for programming languages. In the context of timed UCM specifications, a trace file (saved as a text file) starts with a start point of the root map and terminates with one end point of the root map (in-between, we can have additional start points and end points executions). The AsmL model (Sect. 4.5) is executed within Spec Explorer [18], an advanced model-based specification and conformance testing tool. A generated trace is composed of a sequence of lines recording the name of the UCM construct, its location (i.e. UCM component), current WCET, snapshot of the resource utilization, etc. In addition, the values of some variables of interest can be printed at each computation step.

The inspection (by an analyst) of the generated traces may help discover the following design errors:

– Schedulability issues: This includes responsibilities missing their deadlines, resource exhaustion and end-to-end time constraint violations.
– Lock situations: At the Use Case Maps level of abstraction, no distinction is made between deadlock, livelock and other liveness error situations. A lock situation is detected when the trace does not terminate with an end point that belongs to the root map. Such a lock may be due to incorrect conditions at a branching construct or incorrect plug-in selection policy.
– Violations of user specified correctness assertions: in the context of UCMs, concurrency and non-determinism may impact causality assertions. For instance, analysts may use a generated trace to check for assertions of the following form: *responsibility R2 should always be preceded by responsibility R1*. Furthermore, invariants can be checked by parsing the values of variables at each computation step and computing the invariant expression.
– Violations of postconditions: trace-based testing is efficient to validate that a postcondition holds for every execution of a responsibility that satisfies a precondition. To validate postconditions we need to accomplish the following steps:(1) Identify the execution of the responsibility that is a target for postcondition validation (2) ensure the precondition and (3) validate the postcondition.

5 Illustration Example: Automatic Protection Switching (APS)

Automatic Protection Switching (APS) provides optical protection during times of optical failures in service provider networks. It represents the capability of a transmission system to detect a failure on a working facility and to switch to a standby facility to recover the traffic. This capability has a positive impact on the overall system availability.

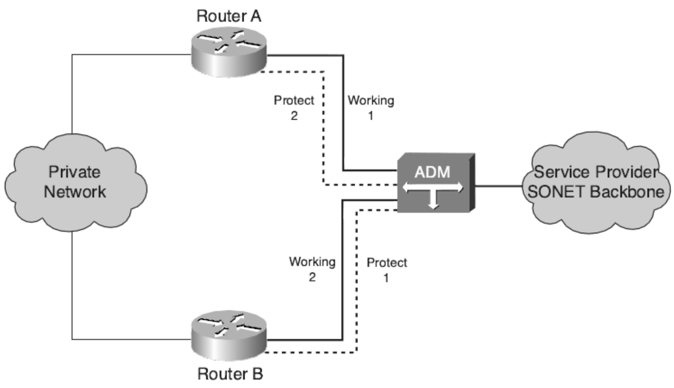


Fig. 10. Typical service provider APS implementation

Telcordia specification GR-253 [19] defines two modes of APS: 1+1 protection switching and 1:N protection switching. In this case study, we limit ourselves to the 1+1 model. In 1+1 protection switching, there is a protection facility (backup line) for each working facility. Figure 10 illustrates a typical service provider SONET (Synchronous Optical NETwork) APS implementation. At the near section (i.e. Add-Drop Multiplexer (ADM)), the optical signal is bridged permanently (split into two signals) and sent over both the working and the protection facilities simultaneously, producing a working signal and a protection signal that are identical. At the far end (i.e. Routers A and B) of the section, both signals are monitored independently for failures.

Protection switching is initiated as a result of one of the following situations:

- Signal Failure (SF) condition: This is a hard failure such as the loss of frame (LOF), Loss of Signal (LOS) within an optical signal.
- Signal Degrade (SD) condition: This is a soft failure caused by the error rate exceeding some predefined value (i.e. SONET line B2 error threshold crossed).
- In response to an external request from a network manager (Manual switch) during a maintenance window.

Two important requirements that every APS implementation should satisfy are:

1. APS must provide the ability to initiate a switch from a failed link within a 10 millisecond time frame.
2. APS must complete the switch from the failed link to a functional link within a 50 millisecond time frame.

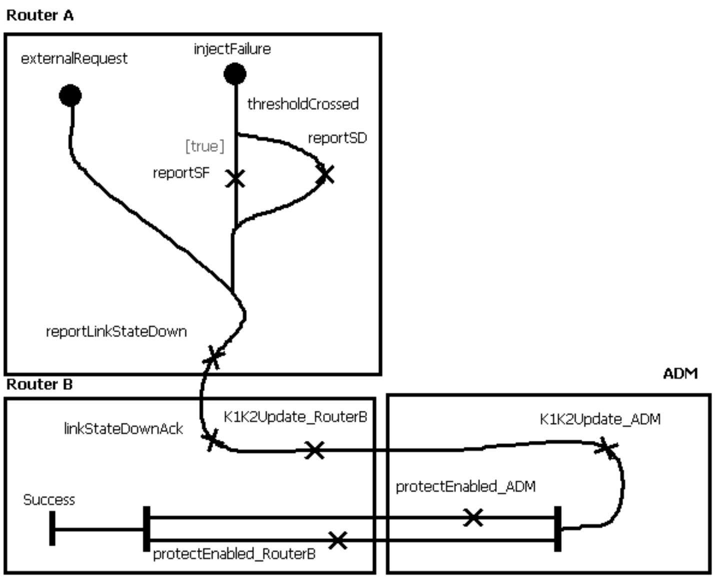


Fig. 11. APS switch UCM scenario

Although simple, this example illustrates the applicability of the proposed approach and covers most of the UCM constructs. Figure 11 provides the timed UCM scenario corresponding to the APS switch mechanism. After reporting a link down state (i.e. responsibility *reportLinkStateDown*), the information exchanged (i.e. responsibility *K1K2Update*) between the router B and the ADM (carried over the K1 and K2 bytes of the SONET overhead) lead to the enabling of the protect link (i.e. responsibility *protectEnabled*).

```
This timed trace is generated with the following initial values:
ThresholdCrossed: true
InjectFailurePreCond : true
externalRequestPreCond: false
```

```
Start Executing APS Switching Scenario
Start Point:injectFailure in component:RouterA; RouterA.RemainingU=90%; WCET=0
OR-Fork: OF1 in component:RouterA; ThresholdCrosses: true; RouterA.RemainingU=85%; WCET=2
Responsibility: reportSF in component:RouterA; RouterA.RemainingU=70%; WCET=7
OR-Join: OJ1 in component:RouterA; RouterA.RemainingU=85%; WCET =8
OR-Join: OJ2 in component:RouterA; RouterA.RemainingU=85%; WCET =9
Responsibility:reportLinkStateDown in component:RouterA; RouterA.RemainingU=70%; WCET=14
Deadline=10 ; reportLinkStateDown missed its deadline
Responsibility:linkStateDownAck in component:RouterB; RouterA.RemainingU=70%; WCET=19
Responsibility:K1K2Updates_RouterB in component:RouterB; RouterB.RemainingU=70%; WCET=24
Responsibility:K1K2Updates_ADM in component:ADM; ADM.RemainingU=70%; WCET=29
AND-Fork: AF in component:ADM; ADM.RemainingU=85%; WCET=30
Responsibility:protectEnabled_ADM in component:ADM; ADM.RemainingU=70%; WCET=35
Responsibility:protectEnabled_RouterB in component:RouterB; RouterB.RemainingU=70%; WCET=40
AND-Join: AJ in component:RouterB; RouterB.RemainingU=85%; WCET=41
End Point: Success in component:RouterB; WCET=41
```

Fig. 12. APS switch timed trace

It is assumed that all constructs are urgent (i.e. delay=0), each UCM control construct (i.e., OR-Fork, OR-Join, AND-Fork and AND-Join) takes 1 time unit to execute and consume 5% of CPU, each responsibility has a duration interval of [2,5] and consumes 20% of CPU. In addition, we assume that all components use 10% of CPU in steady state. Priority is overlooked because no concurrent responsibilities take place within a single component. Figure 12 shows the generated timed trace that corresponds to the APS switch scenario. It indicates that responsibility *reportLinkStateDown* missed its deadline (i.e. 10 *ms*) and that the end-to-end time requirement is met (i.e. WCET = 41 $<$ 50 *ms*).

6 Related Work

Non-functional requirements, in particular time-related aspects, have received considerable attention by the modeling community with several timed extensions of various notations and tools. For instance, the Unified Modeling Language (UML), as a standard, considers real-time aspects in the profile for Schedulability, Performance and Time (SPT) [20]. SPT represents a framework to model resource, time, concurrency, schedulability and performance concepts, to support quantitative analysis of UML models. It defines stereotypes to describe real-time situations (≪SASituation≫), events (≪SATrigger≫), responses (≪SAResponse≫), actions (≪SAAction≫), resources (≪ SAResource≫), processors (≪SAEngine≫), etc. SPT supports schedulability analysis by using ≪SAprofile≫, which uses modifiers on some parameters, such as: (1) worst-case values (as in, *worst-case execution time*), (2) special parameters of a task, such as its release time, its relative and absolute deadlines and laxity (laxity specifies the type of deadline, hard or soft), and (3) special measures such as blocking time and preempted time. More specifically, Di Natali and Saksena [21] have proposed an SPT based schedulability analysis method using Rate Monotonic Analysis (RMA) [7]. For a detailed description of SPT schedulability analysis capabilities and limitations, the reader is invited to consult [22].

The OMG has also recently issued a request for proposal (RFP) for a new UML profile for *Modeling and Analysis of Real-Time and Embedded Systems* (MARTE) [23] in order to upgrade the SPT profile to UML2 [24] and to extend its scope with real-time embedded system (RTES) modeling capabilities. MARTE goes beyond the SPT quantitative model of physical time and adopts more general time models. In MARTE, time can be *physical* (used by chronometric clocks), and considered as *dense* or *discrete*, but it can also be *logical* (i.e., bound to any recurrent event), which focus on the ordering of instants, possibly ignoring the physical duration between instants. In a recent work, Peraldi-Frati and Sorel [25], presented a MARTE-based approach to extract temporal information and the implementation characteristics in order to provide a schedulability analysis. Another working line in this context is the one of Espinoza et al. [26] who provided a framework for MARTE by adopting the modeling practices of the SPT and QoS&FT [27], and proposed a domain model for annotating non-functional properties to support temporal verification of UML based models.

In a closely related work, Petriu et al. [28] have augmented the Use Case Maps language with performance related data such as arrival characteristics for start points, probabilities/weights on branches on OR-fork, plugins and dynamic stubs. The resulting models are then used to generate Layered Queueing Network (LQN) performance models allowing for analysis. However, their approach does not consider explicit delay or execution time of constructs.

Alvarez et al. [29] have proposed a schedulability analysis technique for systems specified in SDL [30]. The authors [29] have defined a predictable execution model that addresses non-determinism (i.e. unpredictable ordering of message and unpredictable process activation) and the interactions with the system resources.

Finally, other research attempts [31,32] integrate the schedulability theory with object-oriented real-time design.

7 Limitations and Future Work

In what follows, some limitations of the scheduling approach and possible remedies are considered:

- *Interleaving vs. true concurrent semantics*: The AsmL implementation of the scheduler, in Sect. 4.5, is based on a priority-driven non-preemptive scheduling with interleaving semantics. This choice is dictated by the lack of true concurrency features in AsmL. To address this limitation, instead of having the system execute a full UCM construct in every single step, the scheduler may be modified to allow the system to make progress on every single clock tick. Although, this solution allows for implementing various scheduling algorithms, it does not scale well. Indeed, simulation times would increase exponentially with responsibilities having large delays and long execution periods.
- *Targeting multiple execution platforms*: For a given timed UCM construct, one single WCET value has been assumed. However, to allow the reuse of UCM specifications across different execution architectures, it is planned to have multiple WCET values attached to a single UCM construct, each one applicable to a given target execution platform.
- *Context dependence of execution times*: At the implementation level, the execution times for individual tasks vary depending on the state of the processor in which they are executed. Tasks using precomputed values (i.e. cached) take less time than expected. This knowledge would allow for a more precise upper bound of execution estimates. In this work, context independence of the timing behavior has been assumed. Such an assumption is acceptable at the UCM level of abstraction.

8 Conclusions

This paper presents a novel approach to perform schedulability analysis at the requirement stage using the Timed Use Case Maps language, extended with time, scheduling and architectural constraints. The resulting specification is formalized in terms of Abstract State Machine (ASM) language and implemented in AsmL allowing for analysis, simulation and validation. The approach has two main benefits. First, it is relatively cheap to implement since it is built upon a previous untimed ASM operational semantics.

Second, it provides an environment (within Spec Explorer) to capture various aspects of a system run (executed constructs, WCET, components, missed deadlines, etc.) in terms of timed traces. This provides the designer, at the early stages, with better understanding of timing properties of the system such as worst-case response times of task, worst-case resource usage, processor utilization, assessment of spare capacity and others.

References

1. Gurevich, Y.: Logic and the Challenge of Computer Science. In: Börger, E. (ed.) Current trends in theoretical computer science, pp. 1–57. Computer Science Press (1988)
2. Hassine, J., Rilling, J., Dssouli, R.: Timed Use Case Maps. In: Gotzhein, R., Reed, R. (eds.) SAM 2006. LNCS, vol. 4320, pp. 99–114. Springer, Heidelberg (2006)
3. International Telecommunications Union: Recommendation Z.151 (11/08), User requirements notation (URN) - Language definition, `http://www.itu.int/rec/T-REC-Z.151/en`
4. Hassine, J., Rilling, J., Dssouli, R.: Abstract Operational Semantics for Use Case Maps. In: Wang, F. (ed.) FORTE 2005. LNCS, vol. 3731, pp. 366–380. Springer, Heidelberg (2005)
5. Hassine, J., Rilling, J., Dssouli, R.: Formal Verification of Use Case Maps with Real Time Extensions. In: Gaudin, E., Najm, E., Reed, R. (eds.) SDL 2007. LNCS, vol. 4745, pp. 225–241. Springer, Heidelberg (2007)
6. Ramamritham, K., Stankovic, J.A.: Scheduling Algorithms and Operating Systems Support for Real-Time Systems. Proceedings of the IEEE 82(1), 55–67 (1994)
7. Liu, C.L., Layland, J.W.: Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. J. ACM 20(1), 46–61 (1973)
8. Ramamritham, K., Stankovic, J.A., Shiah, P.F.: Efficient Scheduling Algorithms for Real-Time Multiprocessor Systems. IEEE Trans. Parallel Distrib. Syst. 1(2), 184–194 (1990)
9. Burns, A., Wellings, A.J.: Real-Time Systems and Programming Languages: ADA 95, Real-Time Java, and Real-Time POSIX. Addison-Wesley Longman, Boston (2001)
10. Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D., Bernat, G., Ferdinand, C., Heckmann, R., Mitra, T., Mueller, F., Puaut, I., Puschner, P., Staschulat, J., Stenström, P.: The Worst-Case Execution-Time Problem—Overview of Methods and Survey of Tools. Trans. on Embedded Computing Sys. 7(3), 1–53 (2008)
11. Nielson, F., Nielson, H.R., Hankin, C.: Principles of Program Analysis. Springer-Verlag New York, Inc, Secaucus (1999)
12. Manna, Z., Pnueli, A.: Clocked Transition Systems. Technical report, Stanford University, Stanford, CA (1996)
13. Alur, R., Dill, D.L.: A theory of Timed Automata. Theor. Comput. Sci. 126(2), 183–235 (1994)
14. Larsen, K.G., Pettersson, P., Yi, W.: UPPAAL in a Nutshell. International Journal on Software Tools for Technology Transfer 1(1-2), 134–152 (1997)
15. Hassine, J.: Formal Semantics and Verification of Use Case Maps. PhD thesis, Concordia University, Montreal, Quebec (2008)
16. Bowman, H., Gomez, R.: Concurrency Theory - Calculi and Automata for Modelling Untimed and Timed Concurrent Systems. Springer, London (2006)
17. ASML: Microsoft Research: The Abstract State Machine Language, `http://research.microsoft.com/en-us/projects/asml/`
18. SpecExplorer: Microsoft Research: Spec Explorer tool, `http://research.microsoft.com/en-us/projects/specexplorer/`

19. Telcordia SONET Transport Systems: GR-253-CORE, Synchronous Optical Network (SONET) Transport Systems: Common Generic Criteria, Section 8, http://telecom-info.telcordia.com/site-cgi/ido/newcust.pl?page=idosearch&docnum=GR-253
20. Object Management Group (OMG): Response to the OMG RFP for Schedulability, Performance and Time, v. 2.0. OMG document ad/2002-03-04 (2002)
21. Di Natale, M., Saksena, M.: Schedulability analysis with UML. In: Lavagno, l., Martin, G., Selic, B. (eds.) UML For Real: Design of Embedded Real-Time Systems, pp. 241–269. Kluwer Academic Publishers, Norwell (2003)
22. Woodside, M., Petriu, D.: Capabilities of the UML Profile for Schedulability Performance and Time (SPT). In: Workshop SIVOES-SPT held in conjunction with the 10th IEEE (2004)
23. Object Management Group (OMG): UML Profile for MARTE, Beta 1, http://www.omg.org/cgi-bin/doc?ptc/07-08-04.pdf
24. Object Management Group (OMG): UML Superstructure Specification, v2.2, http://www.omg.org/docs/formal/09-02-02.pdf
25. Peraldi-Frati, M.A., Sorel, Y.: From High-Level Modelling of Time in MARTE to Real-Time Scheduling Analysis, http://www-rocq.inria.fr/syndex/pub/aces08/aces08.pdf
26. Espinoza, H., Dubois, H., Gérard, S., Pasaje, J.L.M., Petriu, D.C., Woodside, C.M.: Annotating UML models with non-functional properties for quantitative analysis. In: Bruel, J.-M. (ed.) MoDELS 2005. LNCS, vol. 3844, pp. 79–90. Springer, Heidelberg (2006)
27. Object Management Group (OMG): UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms, http://www.omg.org/docs/formal/06-05-02.pdf
28. Petriu, D.C., Amyot, D., Woodside, C.M.: Scenario-Based Performance Engineering with UCMNAV. In: Reed, R., Reed, J. (eds.) SDL 2003. LNCS, vol. 2708, pp. 18–35. Springer, Heidelberg (2003)
29. Alvarez, J.M., Diaz, M., Llopis, L., Pimentel, E., Troya, J.M.: Integrating Schedulability Analysis and Design Techniques in SDL. Real-Time Syst. 24(3), 267–302 (2003)
30. International Telecommunications Union: Recommendation Z.100 (11/07), Specification and Description Language (SDL), http://www.itu.int/rec/T-REC-Z.100/en
31. Kim, S., Cho, S., Hong, S.: Automatic Implementation of Real-Time Object-Oriented Models and Schedulability Issues. In: Proceedings of the Sixth International Workshop on Object-Oriented Real-Time Dependable Systems, pp. 137–141. IEEE Computer Society Press, Los Alamitos (2001)
32. Saksena, M., Karvelas, P., Wang, Y.: Automatic Synthesis of Multi-Tasking Implementations from Real-Time Object-Oriented Models. In: ISORC 2000: Proceedings of the Third IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, pp. 360–367. IEEE Computer Society Press, Los Alamitos (2000)

Extending the User Requirements Notation with Aspect-Oriented Concepts

Gunter Mussbacher and Daniel Amyot

SITE, University of Ottawa, 800 King Edward, Ottawa, ON, K1N 6N5, Canada
{gunterm,damyot}@site.uottawa.ca

Abstract. In November 2008, the User Requirements Notation (URN) was approved as Recommendation Z.151 by the Standardization Sector of the International Telecommunication Union (ITU-T). URN is the first and currently only standard that supports both goal-oriented and scenario-based modeling for requirements engineering activities. The Aspect-oriented URN (AoURN) is a recent extension of URN that combines goals, scenarios, and aspects in one framework. AoURN is a candidate for future versions of the URN standard. We first summarize the basic concepts and notation of AoURN and then discuss advanced features of AoURN that are necessary for large-scale modeling. Based on our experience with AoURN modeling, we present a list of requirements including rationale for an aspect-oriented extension of Z.151 that succinctly expresses the required features to evolve URN into a complete aspect-oriented modeling framework.

Keywords: User Requirements Notation, Goal-oriented Requirement Language, Use Case Maps, Aspects, Aspect-oriented Modeling, Aspect-oriented Requirements Engineering, Aspect-oriented User Requirements Notation.

1 Introduction

The recent Recommendation Z.151 (11/08) defines the User Requirements Notation (URN) [9], a modeling language for requirements engineering and high-level design that incorporates goal-oriented and scenario-based models in one framework. While scenario models have been regarded as an essential tool for software development for a long time, goal models are a more recent development [5,22,26,28]. In URN, goal models created with the Goal-oriented Requirement Language (GRL) are complemented with scenario models created with Use Case Maps (UCMs). Even though Recommendation Z.151 is now available, new developments in software engineering must be considered in the context of evolving URN. One of the most exciting developments in software engineering over the last decade is the emergence of aspect-oriented modeling (AOM) [4], promising better encapsulation of crosscutting concerns at the requirements and architecture stage which in turn may lead to greater maintainability, reusability, and scalability of requirements models. Crosscutting concerns in the context of requirements engineering are for example features, scenarios, and non-functional requirements (NFRs) that cannot be properly encapsulated with only traditional requirements engineering techniques. Many aspect-oriented approaches for such techniques have been proposed (e.g., for use cases [2,3,10], viewpoints [24], problem frames [12], and UML models [6,27]).

For the last three years, the Aspect-oriented User Requirements Notation (AoURN) has been developed [13,14,15,16,17,18,19,20,21,23]. During this time period, the initial basic concepts of AoURN have been augmented with several advanced features. We summarize all of these advanced features together here in one publication for the first time, and furthermore discuss the motivation of these advanced features. Based on our experiences in combining URN and aspect-oriented concepts, numerous case studies, and feedback from industrial collaborators, we are now in a position to formulate requirements for the extension of URN with aspects. This approach follows the example of URN itself, as requirements for URN were first published in Recommendation Z.150 (02/03) "User Requirements Notation (URN) – Language requirements and framework" [8] and later Z.151 was defined based on these requirements. The goal therefore is to amend Z.150 with requirements for aspect-oriented modeling based on the requirements presented here. Furthermore, Z.151 itself contains additional requirements that describe in detail the semantics of the dynamic behavior of UCMs when interpreted (i.e., traversed) by a path traversal mechanism. Hence, the second set of requirements presented here is formulated for the UCM path traversal to enable the traversal of aspect-oriented UCM models.

In the remainder of this paper, Section 2 gives a brief overview of Recommendation Z.151, the User Requirements Notation (URN). Section 3 explains the basic concepts of the Aspect-oriented User Requirements Notation (AoURN). Section 4 then addresses advanced features of AoURN required for large-scale modeling. A simple web-based application serves as an example to illustrate the basic and motivate the advanced features of AoURN. Section 5 presents the list of requirements for extending URN with aspect-oriented concepts, grouped into general requirements and requirements for the path traversal of Use Case Maps (UCM). Finally, Section 6 concludes the paper and identifies future work.

2 Overview of Z.151 – User Requirements Notation

The User Requirements Notation (URN) [1,9] supports the elicitation, analysis, specification, and validation of requirements. URN captures early requirements in a modeling framework containing two complementary sub-languages called Goal-oriented Requirement Language (GRL – for goal-oriented modeling) and Use Case Maps (UCMs – for scenario-based modeling). GRL models are used to describe and reason about non-functional requirements (NFRs), quality attributes, and the intentions of system stakeholders, whereas UCM models are used for operational requirements, functional requirements, and performance and architectural reasoning. While GRL identifies at a very high level of abstraction possible solutions to be considered for the proposed system, UCM models describe these solutions in more detail. In summary, URN has concepts for the specification of stakeholders, goals, non-functional requirements, rationales, behaviour, actors, scenarios, and structuring.

A GRL model consists of *intentional elements* (e.g., softgoals (⬭), hard goals (⬭), and tasks (⬡)) connected together with different types of links. Intentional elements may be assigned to stakeholders called *actors* (⬭). *Contribution links* (⟶) indicate positive (+) or negative (–) impact of intentional elements on each other. *Correlation links* (⇢) are similar to contribution links in that they also indicate impact but are used to describe side effects rather than desired impacts. *Decomposition links* (+—) allow the decomposition of intentional elements into sub-elements. AND, (inclusive) OR, and XOR decompositions are supported.

For example in Fig. 1, two stakeholders, the Customer and the DVD Store, are shown. The customer's goal graph has three intentional elements with positive contribution links between them, whereas the store's goal graph is a little more complicated and considers two alternatives, Online Store and Traditional Store, modeled as tasks.

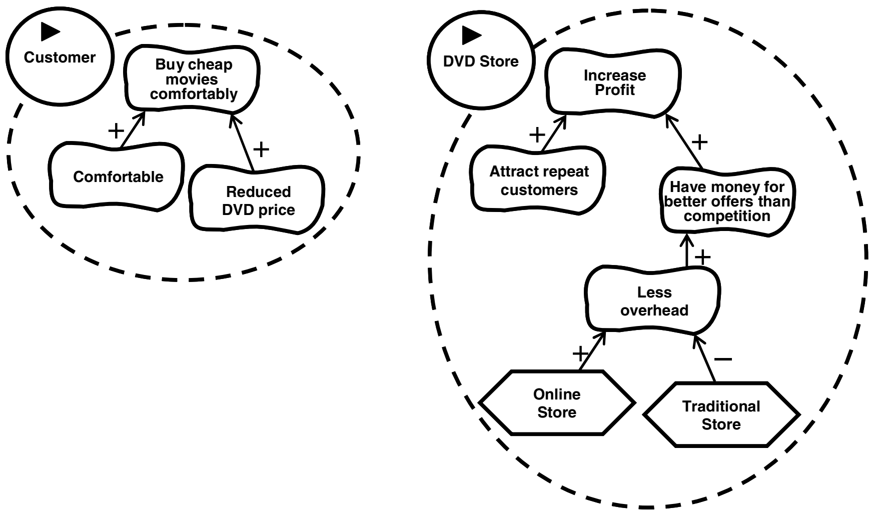


Fig. 1. The Customer and DVD Store Stakeholders of a Simple Online DVD Store System

A UCM model (such as Fig. 2) consists of a path that begins at a *start point* (●, e.g., buy) and ends with an *end point* (▌, e.g., bought). A path may contain *responsibilities* (×, e.g., processOrder), identifying the steps in a scenario, and notational symbols for alternative (—<) and concurrent (—|=) branches. Path elements may be assigned to a *component* (□, e.g., DVD Store). *Stubs* (◇) are containers for sub-models called *plug-in maps*. Drilling into a stub leads to a submap that provides more details, thus allowing for hierarchical structuring of UCM models. A binding between the stub and elements on the plug-in map precisely defines how the scenario continues from the parent map to the submap and back to the parent map.

Furthermore, *URN links* (▶) are used to establish traceability by relating tasks (i.e., possible solutions) or actors in goal models to their representation in the UCM model (e.g., maps, responsibilities, and stubs or components that further describe the linked GRL elements). For example, the GRL Customer stakeholder in Fig. 1 is linked to the UCM Customer component in Fig. 2. Finally, URN allows the definition of *metadata* (<<MetadataName>>) for any URN modeling element, thus providing an extension mechanism that permits the definition of profiles for URN.

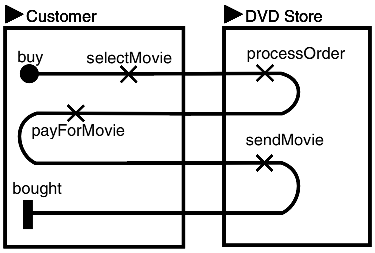


Fig. 2. The Buy Movie Use Case of a Simple Online DVD Store System

The most comprehensive URN tool available to date is the Eclipse plug-in jUCMNav [11]. jUCMNav is a full editor for GRL and UCM models that ensures that only syntactically correct URN models are created. The tool manages hierarchical UCM models consisting of several layers of maps and plug-in maps and allows the requirements engineer to navigate easily through them. In addition, the tool provides standard analysis features for URN models: global quantitative and qualitative tradeoff analysis for conflicting stakeholder goal models as well as validation and (regression) testing of UCM models including the detection of undesired feature interactions. jUCMNav supports the built-in traceability and profiling features of URN as well as several transformations. Structured, textual use cases may be imported into URN models and URN models may be transformed into more detailed scenario languages such as message sequence charts but also into performance models. Furthermore, jUCMNav synchronizes URN models with requirements management tools and therefore allows URN models to be managed together with other types of requirements. OCL constraints on URN models can also be defined and verified by jUCMNav. Some support for aspect-oriented modeling is already available for jUCMNav. Further AO functionality is being prototyped and will be added to the tool in the near future. Finally, the tool also supports more advanced research on URN-based business process monitoring and runtime adaptation. For more details about URN, visit the URN Virtual Library [25].

3 Basic Concepts of AoURN

The Aspect-oriented User Requirements Notation (AoURN) [15,16,17,18,19,21,23] extends the User Requirements Notation (URN) with aspect-oriented concepts, allowing modelers to better encapsulate crosscutting concerns which are hard or impossible to encapsulate with URN models alone. AoURN adds aspect concepts to URN's sub-languages, leading to and integrating Aspect-oriented GRL (AoGRL) [17,23] and Aspect-oriented UCMs (AoUCM) [16,18,19]. The three major aspect-oriented concepts that have to be added to URN are concerns, composition rules, and pointcut expressions. Note that the term aspect refers to a crosscutting concern, while the term concern encompasses both crosscutting and non-crosscutting concerns. These are core concepts of many aspect-oriented modeling (AOM) techniques. In terms of aspect-oriented programming (AOP, e.g. with AspectJ [7]), the concept of a crosscutting concern in AOM relates to the concept of an aspect and the aspect's advice in AOP, the concept of composition rules in AOM encompasses the common before/after/around operators in AOP, and the concept of pointcut expressions is the same in AOM and AOP.

A *concern* is a new unit of encapsulation that captures everything related to a particular idea, feature, quality, etc. AoURN treats concerns as first-class modeling elements, regardless of whether they are crosscutting or not. Typical concerns in the context of URN are stakeholders' intentions, NFRs, and use cases. AoURN groups all relevant properties of a concern such as goals, behavior, and structure, as well as pointcut expressions needed to apply new goal and scenario elements to a URN model or to modify existing elements in the URN model.

Pointcut expressions are patterns that are specified by an aspect and matched in the URN model (often referred to as the base model). If a match is found, the aspect is applied at the matched location in the base model. The *composition rule* defines how an aspect transforms the matched location. AoURN uses standard URN diagrams to describe pointcut expressions and composition rules (therefore AoURN is only limited by the expressive power of URN itself as opposed to a particular composition language). AoURN's aspect composition technique can fully transform URN models.

Section 2 has already introduced a URN model with three concerns for the example application: the Customer stakeholder, the DVD Store stakeholder, and the Buy Movie use case. These concerns will now be expanded and new concerns will be added in the remainder of this section with the help of basic AoURN features.

AoURN adds the ability to define pointcut expressions and then compose aspects with the URN model. GRL pointcut expressions are shown on a *pointcut graph* and make use of *pointcut markers* (P) and *pointcut deletion markers* (×) to indicate the pattern to be matched. All elements without pointcut markers are added to the matched location in the GRL base model, while elements with a pointcut deletion marker are removed. The composition rule is therefore defined by the set of links between elements without pointcut markers and elements with pointcut markers. Generic, reusable goals and tasks of an aspect may be described in more detail in separate goal graphs called *aspect graphs*.

For example, Fig. 3 depicts four pointcut graphs showing how the DVD Store and the Customer stakeholders are connected (Fig. 3.a), how the Buy Movie use case and the new Earn/Redeem Movie Points use case impact the goal model (Fig. 3.b and Fig. 3.c), and how the new Security NFR concern impacts the use cases (Fig. 3.d). Note that the tasks defined on the pointcut graphs of the use case concerns have URN links to UCM elements (e.g., Process online and send by mail is linked to processOrder and sendMovie shown in Fig. 2). Furthermore, a reusable aspect graph of the Security NFR concern is shown in Fig. 4, explaining how security can be generally achieved by modeling knowledge from the security domain.

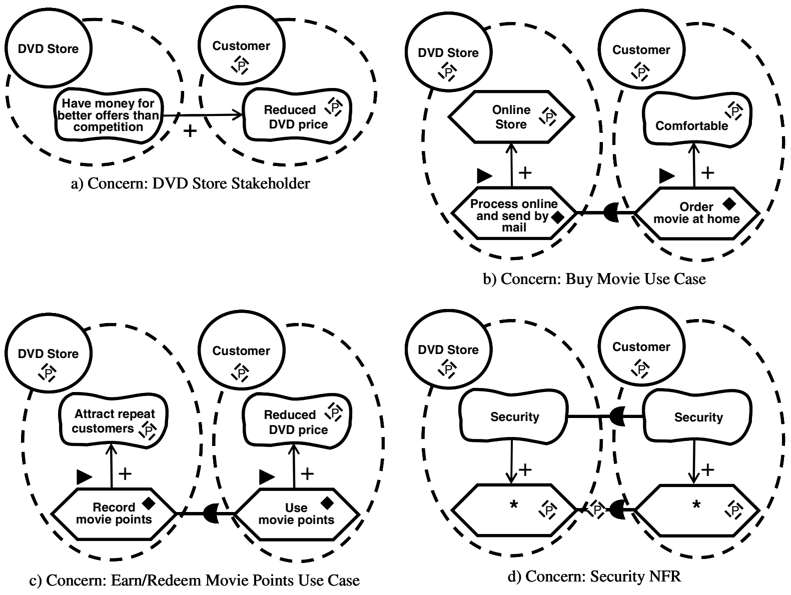


Fig. 3. AoGRL's Pointcut Graphs for a Simple Online DVD Store

The pointcut expressions in Fig. 3.a, Fig. 3.b, and Fig. 3.c are rather straightforward and the composition of these aspects with the base model is simple. The base locations affected by these three aspects are indicated by four *aspect markers* (◆, e.g., Reduced DVD price) in Fig. 5. The pointcut expression of the Security NFR concern in Fig. 3.d connects the reusable aspect graph of the Security NFR concern with the application-specific goal model of the online DVD store. The pointcut expression matches any two tasks with a dependency link as long as one task resides within the DVD Store stakeholder while the other task resides within the Customer stakeholder. The base locations affected by the Security NFR concern are again indicated by aspect markers (see the four tasks in Fig. 3.b and Fig. 3.c).

When an aspect marker is selected, the modeler is taken to the *AoView* of the aspect with only those aspectual properties of the pointcut graph highlighted

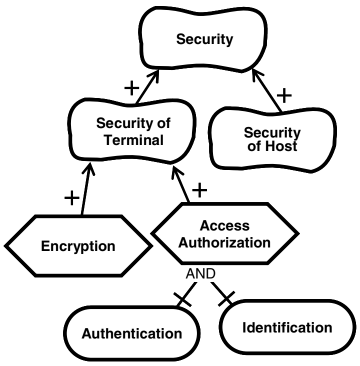


Fig. 4. AoGRL's Aspect Graph for the Security NFR Concern

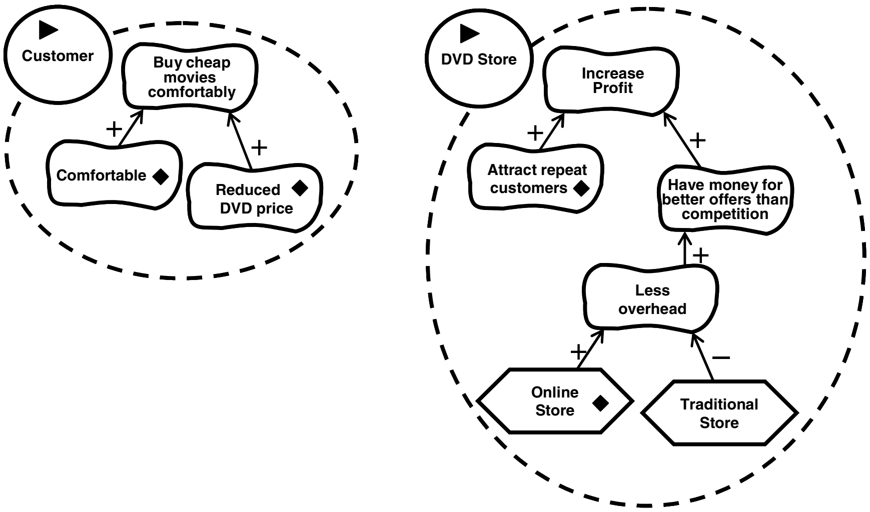


Fig. 5. AoGRL's Aspect Markers for a Simple Online DVD Store System

that are relevant to the aspect marker. The aspect markers and the AoViews allow the requirements engineer to reason about the composed model. The concept of composed models in AOM thus relates to aspects being woven into the base in AOP. Essentially, aspect markers in AoURN are similar to the advice markers in AspectJ shown on the left-hand side of the Java Eclipse editor [7].

The AoViews of the aspect markers in Fig. 5 are simply the pointcut graphs from Fig. 3 without the symbols for the pointcut markers – i.e., the AoView of the aspect marker for Comfortable is Fig. 3.b, for Reduced DVD price it is Fig. 3.a and Fig. 3.c because Reduced DVD price is matched by two pointcut graphs, for Attract repeat customers it is Fig. 3.c (see Fig. 6.a), and for Online Store it is Fig. 3.b. The two examples illustrated in Fig. 6.b and Fig. 6.c show the AoViews

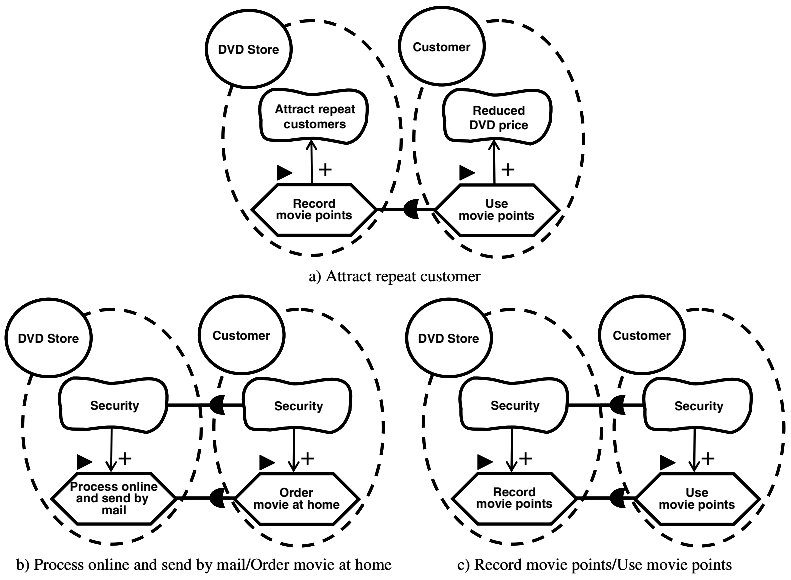


Fig. 6. AoGRL's AoView for five Aspect Markers

corresponding to the aspect markers of the Security NFR. The AoView of the aspect markers of Order movie at home and Process online and send by mail is shown in Fig. 6.b, while the AoView of the aspect markers of Record movie points and Use movie points is shown in Fig. 6.c. Note how parameterized elements of the pointcut expression are replaced by their actual matches in the AoViews in Fig. 6.b and Fig. 6.c. By using the models defined by the requirements engineer for the AoViews, it is possible to view the composed AoURN model without having to resolve complex layout issues (this applies to AoGRL and AoUCM).

Similarly to GRL pointcut expressions, UCM pointcut expressions define the pattern to be matched with a *pointcut map*. Grey start and end points on the pointcut map are not part of the pointcut expression but rather denote its beginning and end. The aspectual properties are shown on a separate *aspect map*, allowing the pointcut expression and the aspectual properties to be individually reused. The aspect map is linked to the pointcut expression with the help of a *pointcut stub* (◇). The causal relationship of the pointcut stub and the aspectual properties visually defines the composition rule for the aspect, indicating how the aspect is inserted in the base model (e.g., before, after, optionally, in parallel or anything else that can be expressed with the UCM notation). The *replacement pointcut stub* (◆) is a special kind of pointcut stub, indicating that the aspect is replacing the matched base elements.

For example, the purpose of the new Communication concern in Fig. 7 is to define in more detail the interaction between the customer and the online DVD store when the customer selects a movie. The pointcut map therefore matches against the selectMovie and processOrder responsibilities in the Customer and

DVD Store components, respectively. The bindings of the pointcut stub connect the in-path of the stub with the start point of the pointcut map and the out-path of the stub with the end point of the pointcut map. As the pointcut stub on the aspect map is a replacement pointcut stub, the matched responsibilities are replaced with the aspectual properties described on the aspect map. The aspect map defines that selectMovie and processOrder are reinserted and explicit request and reply responsibilities as well as a waiting place are added, specifying that the customer has to wait for the response of the online DVD store. Note that the specification of this aspect is rather problematic as will be discussed in Sect. 4. However, it serves its purpose here to introduce the main concepts of AoUCM.

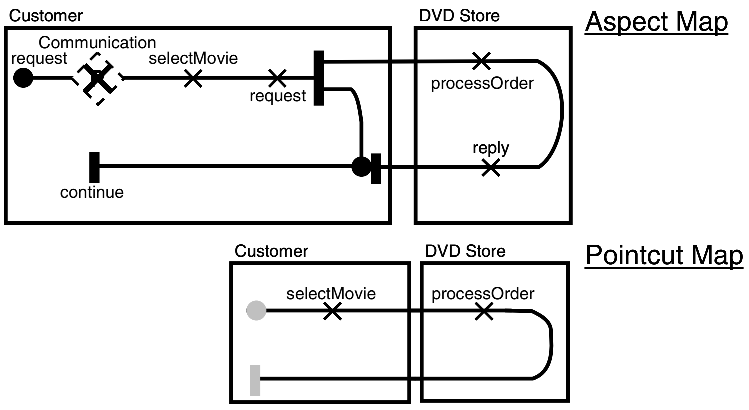


Fig. 7. AoUCM's Aspect Map and Pointcut Map for the Communication Concern

Similar to GRL aspect markers, UCM aspect markers (◆) also indicate affected base locations in the UCM model. If the aspect adds elements before or after the base location matched by the pointcut expression, the aspect marker is added before or after the base location, respectively. In the case of a replacement, two aspect markers are added, one before and one after the replaced base elements. In contrast to AoGRL, a UCM aspect marker is not just an annotation but is rather a kind of stub that links the base model with a submap: the aspect map. Bindings between the aspect marker and the aspect map are created automatically by AoURN's composition mechanism. Figure 8 shows the aspect markers added to the use case from Fig. 2 because the pointcut expression in Fig. 7 matches selectMovie and processOrder in this use case.

Fig. 8 also shows the AoView for the aspect markers, highlighting the portion of the Communication aspect map that is inserted. When the first aspect marker is reached during the Buy Movie scenario, the scenario continues with the aspectual behavior on the aspect map (right after the pointcut stub). When the aspect map's end point is reached, the scenario continues with the second aspect marker, thus skipping the replaced base elements. If the aspect does not replace base elements but simply adds elements before or after the matched base location,

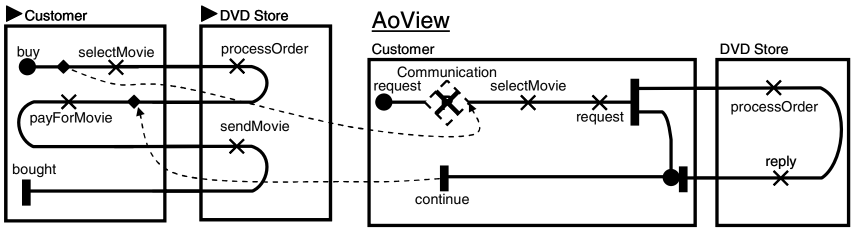


Fig. 8. AoUCM's Aspect Markers and their corresponding AoView

the scenario returns from the aspect map to the same aspect marker. In this case, the aspect marker has bindings to and from the aspect map. Note that the dashed arrows depicting bindings are only added for illustration purposes and are not part of AoUCM's concrete syntax. The jUCMNav tool manages bindings and hierarchical UCM models much more effectively.

Finally, aspects may depend on or conflict with each other. AoURN models dependencies and conflicts among concerns and the resolution thereof with the help of the *Concern Interaction Graph* (CIG) [15]. Many aspect interactions can be resolved with precedence rules. The CIG is a specialized GRL goal graph that uses dependencies, correlations, and intentional elements to model such precedence rules. The precedence rules then govern the order in which concerns are applied to a URN model.

4 Advanced Features of AoURN

The pointcut expression in Fig. 7 is problematic because only a very specific interaction between the customer and the online DVD store is matched. However, it is very likely that many if not all interactions between the customer and the online DVD store use the request/reply pattern defined by the aspect in Fig. 7. To rectify this problem, the pointcut expression can easily be adapted to match against any two responsibilities (simply by using the wildcard * instead of concrete names for the responsibilities).

This is better but still very fragile with respect to rather small changes to the base model. If another responsibility is added in the base model after processOrder (e.g., rewardReferrer), then the pointcut expression will no longer match. The pointcut expression expects only two responsibilities before the path crosses back into the Customer component. Therefore, the pointcut expression must be made more flexible. This is achieved by the *anything pointcut element* (.....) as shown in Fig. 9. The anything pointcut element matches against an arbitrary sequence of base elements.

At this point, the aspect itself is not generic enough and therefore cannot be reused easily, because it also specifies very concrete elements of the base model (i.e., Customer, DVD Store, selectMovie, processOrder). In order to model a reusable aspect, it must be possible to reuse base elements matched by the

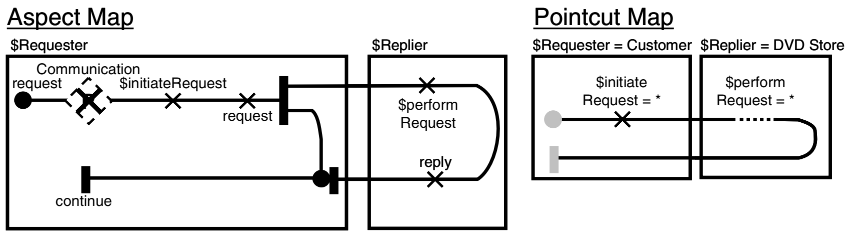


Fig. 9. Reusing Matched Elements with Variables and the Anything Pointcut Element

pointcut expression. This is achieved with *variables* denoted by the prefix $ as shown in Fig. 9 on the aspect map and pointcut map.

The pointcut expression in Fig. 9 now matches against selectMovie, processOrder, and rewardReferrer as well as payForMovie and sendMovie (hence there are two pairs of aspect markers in Fig. 10). Furthermore, when the aspect is applied to a location, the variables in the aspect are replaced by the matched elements (i.e., $initiateRequest is replaced by selectMovie or by payForMovie; $performRequest is replaced by processOrder and rewardReferrer or by sendMovie; $Requester is replaced by Customer; and $Replier is replaced by DVD Store as shown in the AoViews in Fig. 10).

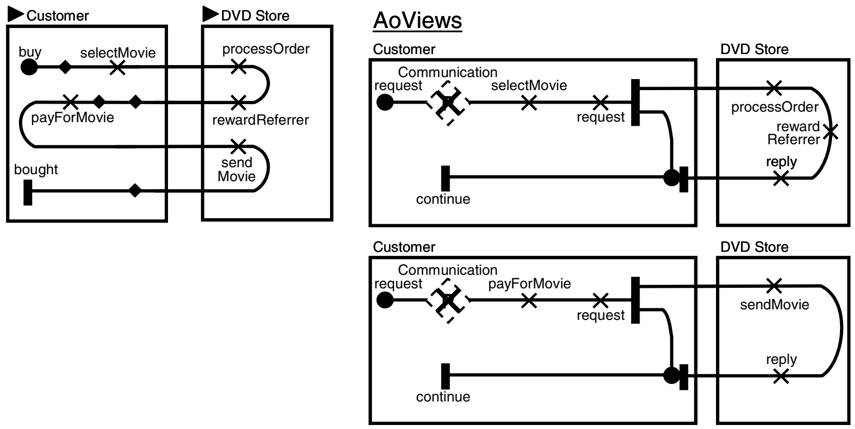


Fig. 10. AoUCM's Aspect Markers and their corresponding AoView

Note that the anything pointcut element is only available for AoUCM. It is not available for AoGRL because the highly interconnected nature of goal graphs leads to an undesired explosion of matches. Furthermore, AoGRL does not require variables because GRL pointcut expressions are not separated from the description of aspectual properties. Although this is an advantage, the disadvantage is that individual reuse of pointcut expressions and aspectual behavior, while possible with AoUCM, is not possible with AoGRL.

The pointcut expression in Fig. 9 now captures all interactions between the customer and the DVD store. However, some interactions may have to be treated differently, e.g., asynchronous communication may be more appropriate for some interactions. This may lead to pointcuts with complicated regular expressions. Annotation-based matching is a common approach for aspect-oriented programming languages and can be effectively used to address this issue. AoURN supports *annotation-based matching* with metadata for both AoGRL and AoUCM models. Metadata may be specified in the pointcut expression and the base model may also be annotated with the metadata, explicitly identifying the locations to which the aspect should be applied. The metadata is then simply taken into account by the matching algorithm of AoURN.

The aspect map in Fig. 9, which describes a communication mechanism, may be further improved by adding checks for corrupted replies and a way to retry the interaction. Figure 11 shows the updated aspect map. This, however, leads to an ambiguous situation when the aspect map is connected to the aspect marker through bindings. It is not clear which of the two start points should be connected and which of the two end points. This is addressed by the definition of *local start and end points* (⦸ and ⍂, e.g., retry and fail in Fig. 11), which by definition are never connected by bindings with aspect markers.

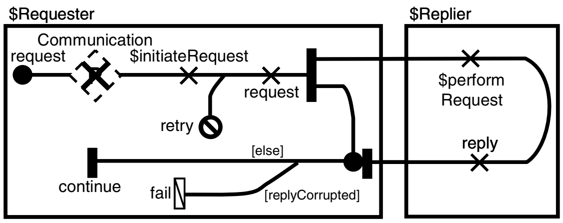


Fig. 11. Local Start and End Points for Aspects

An attempt to describe in more detail the Earn/Redeem Movie Points use case introduced in Fig. 3 uncovers another problem. The Buy Movie (BM) use case and the Earn/Redeem Movie Points (ERMP) use case are heavily intertwined with each other. First a membership form needs to be filled (ERMP), then the order is processed (BM), then movie points may be redeemed if enough points are available (ERMP). If there are not enough points available (ERMP), then the customer pays for the movie as usual (BM) but earns movie points (ERMP) once the movie has been sent (BM). The problem with intertwined use cases is that either both use cases have to be modeled together (i.e., different concerns are tangled with each other) or many small aspects add the individual steps of one use case to the other, making it hard to understand how these small aspects relate to each other. Intertwined use cases are a very common phenomenon. AoUCM uses *interleaved composition* to address this problem by allowing multiple pointcut stubs to be defined for a single aspect map.

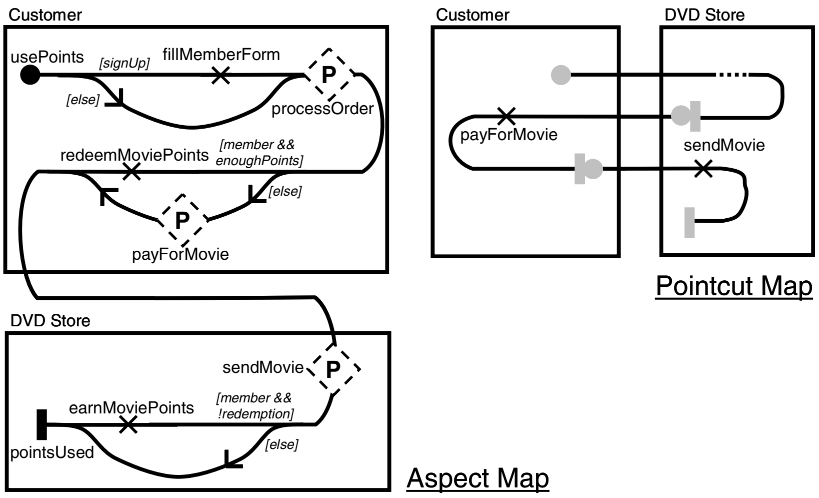


Fig. 12. Interleaved Composition

A requirements engineer can understand the ERMP use case simply by looking at its definition, i.e., the aspect map in Fig. 12. First, a member form must be filled if the customer is not yet signed up, then movie points may be redeemed if enough points are available, and finally movie points may be earned if the member did not redeem movie points in the same transaction. In addition, the pointcut stubs identify how the BM use case is interleaved with the ERMP use case. The member form must be filled before processing the order, redeeming of movie points may occur instead of paying for a movie, and earning movie points happens after the movie was sent.

As shown in the pointcut expression in Fig. 12, interleaved composition is achieved with a series of UCM paths, established by connecting together grey end and start points. Pairs of grey end and start points are ignored by the matching algorithm and the pointcut expression is therefore easily matched against the Buy Movie use case described in Fig. 10. The additional grey end and start points, however, are used when connecting a series of pointcut stubs to a series of UCM paths on the pointcut map. All examples until now featured only pointcut maps with one grey start and one grey end point and aspect maps with only one pointcut stub. The binding to the pointcut stub on the aspect map was therefore straightforward – the in-path of the pointcut stub is connected to the grey start point and the out-path to the grey end point. In Fig. 12, the processOrder pointcut stub is connected to the first path segment of the pointcut expression, the payForMovie pointcut stub to the second path segment, and the sendMovie pointcut stub to the third and last path segment. Note how the causal relationship of the individual steps of the Earn/Redeem Movie Points use case are clearly described on the aspect map and how the bindings of pointcut stubs to UCM paths define the interleaving of the two use cases, while the pointcut

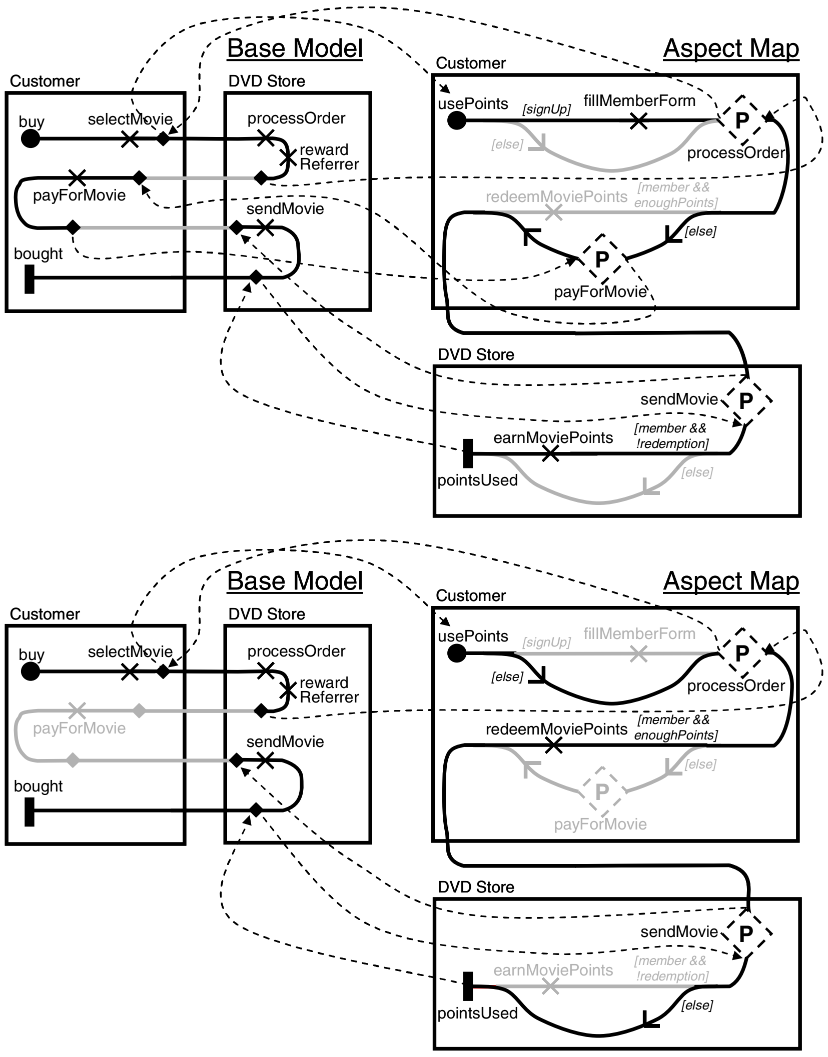


Fig. 13. Two Highlighted Scenarios of the Earn/Redeem Movie Points Use Case

expression remains fairly simple. Interleaved composition is powerful yet it is seldom supported in other AOM and AOP approaches.

Fig. 13 highlights two scenarios of the Earn/Redeem Movie Points use case composed with the Buy Movie use case. The first is the initial scenario where a new member signs up to the movie points program, while the second scenario shows a redemption of movie points. The dashed arrows have been added to illustrate the bindings created automatically by the composition mechanism of AoURN, allowing the reader to follow along the scenario as it unfolds. Note that the jUCMNav tool does not require the bindings to be visualized but allows the modeler to conveniently navigate from stubs to plug-in maps or aspect markers to aspect maps by clicking on the stub or aspect marker, respectively. This results in a much less confusing experience for the requirements engineer than Fig. 13 seems to convey.

The operationalization of the Security NFR concern from Fig. 3.d can also be modeled concisely with interleaved composition. The pointcut expression matches against the request responsibility, the AND-fork, $performRequest, reply, and the path crossing back into the $Requester component in the Communications concern. Encryption processing is then added after the request and reply responsibilities, while decryption processing is added before $performRequest and after crossing back into the $Requester component.

Some of the aspect markers in Fig. 13 only have a binding to the aspect map, some only one from the aspect map, and some both. The concrete syntax for all three cases, however, is the same. This makes it difficult for the requirements engineer to understand the impact of an aspect simply by looking at the aspect marker. AoUCM differentiates between these three cases (◆ . . . standard = to and from bindings, ◆▶ . . . beginning of a replacement = only to bindings, ▶◆ . . . end of a replacement = only from bindings), making it possible to *identify replacements immediately*.

5 Requirements for Extending Z.151 with Aspects

Table 1 proposes requirements for improving modularity and separation of concerns in URN by extending it with aspect-oriented concepts. Some of these requirements may not require any changes to the current version of URN, especially if a proposed solution for AOM with URN uses existing modeling constructs specialized by metadata in a URN profile. These requirements are still included here as this set of requirements aims to be complete with respect to coverage of key aspect-oriented concepts.

6 Conclusion

We have presented the Aspect-oriented User Requirements Notation (AoURN), an extension to the User Requirements Notation (URN). Basic and advanced features of AoURN were illustrated and discussed. Based on these features as well as our experience with AoURN modeling in general, we proposed a list of requirements that can be used to evolve URN into a complete aspect-oriented modeling environment for requirements engineering activities. In future work, the requirements for supporting the analysis and validation features of URN in an aspect-oriented way as well as the requirements for the path traversal mechanism will still have to be refined. Furthermore, recent work on analyzing semantic aspect interactions with the help of GRL graphs could be considered for inclusion into an updated version of the URN standard. Finally, the notions of aspect markers and pointcut expressions spanning model types with the help of URN links could be applied to other AOM techniques to address layout issues of composed models and support multi-model aspects.

Table 1. Requirements for Extending Z.151 with Aspect-oriented Concepts

Requirement (URN shall allow. . .)	**Rationale**
GRL and UCM model diagrams and elements to be grouped into concerns.	Basic premise of aspect-oriented modeling.
Patterns (i.e., pointcut expressions) to be defined for a concern.	Basic premise of aspect-oriented modeling.
Patterns to be defined that are at the same level of complexity as URN itself.	A pattern language (i.e., pointcut language) that is not able to capture all constructs of URN is limiting the expressiveness and flexibility available to the requirements engineer.
Parameterized patterns to be defined.	Parameterized patterns allow concerns to transform a large numbers of base locations.
Patterns that include logical operators AND, OR, or NOT.	Required to fine-tune pattern descriptions.
Patterns that include metadata.	Required to fine-tune pattern descriptions.
Patterns that match against an arbitrary sequence of UCM base elements.	Required to safe-guard successful pattern matches against small changes in the base model or to efficiently capture variations in the target pattern.
The reuse of matched UCM elements in the description of the concern properties.	Required to specify highly reusable concerns.
Patterns to be defined over both model types.	Aspect-oriented extensions to URN must take advantage of the URN framework.
Purely syntactic model elements to be ignored when matching the pattern against the base model.	Required to safe-guard successful pattern matches against inconsequential changes in the base model.
Semantically equivalent models to be matched even if their syntactic representation is different.	Required to safe-guard successful pattern matches against inconsequential changes in the base model.
Composition rules to be defined for a concern.	Basic premise of aspect-oriented modeling.
Composition rules to be defined that are at the same level of complexity as URN itself.	Composition rules that are not able to express all constructs of URN are limiting the expressiveness and flexibility available to the requirements engineer.
Interleaved composition to be defined.	Scenarios are often intertwined with each other. Interleaved composition allows scenarios to be defined separately from each other without losing context information and increasing pattern complexity.
Locations to be transformed by a concern only if the transformation is indicated at the locations.	Basic feature of aspect-oriented modeling and programming.
Locations to be transformed by a replacement transformation of a concern only if the replacement transformation is indicated at the locations in a different way than for locations that are transformed in some other way.	It is important for the requirements engineer to understand at a glance the impact of concerns on the matched base locations.
Concern dependencies and conflicts as well as their resolutions to be modeled.	Concerns interact with each other. Without support for the resolution of these interactions, the model specification is ambiguous and may result in undesired interactions.
The individual, separate reuse of concern properties and patterns (patterns may only be reused for UCM but not for GRL).	A goal of aspect-oriented modeling is to provide highly reusable assets. URN must support this by allowing concern properties and patterns to be reused separately from each other.
The base model composed with all concerns to be viewed without requiring the user to resolve complex layout issues.	Requirements engineers need to have access to a fully composed model for further analysis. URN must support this.
Its analysis and validation features to be performed in an aspect-oriented way.	This is a very general requirement at this point. However, any aspect-oriented extension to URN must not prevent or limit the current analysis and validation features.
The UCM Path Traversal to traverse the specified concern behavior upon arrival at a location transformed by a concern before continuing with the traversal at a base location.	This is also quite general at this point and will have to be refined given a particular aspect-oriented extension of URN.

Acknowledgments

This research was supported by the Natural Sciences and Engineering Research Council of Canada, through its programs of Discovery Grants and Postgraduate Scholarships.

References

1. Amyot, D., Mussbacher, G.: Development of Telecommunications Standards and Services with the User Requirements Notation. Workshop on ITU System Design Languages (2008), http://www.itu.int/dms_pub/itu-t/oth/06/18/T06180000010012PDFE.pdf
2. Araújo, J., Coutinho, P.: Identifying Aspectual Use Cases Using a Viewpoint-Oriented Requirements Method. In: Early Aspects 2003: Aspect-Oriented Requirements Engineering and Architecture Design, Workshop of the 2nd International Conference on Aspect-Oriented Software Development, Boston (2003)
3. Araújo, J., Moreira, A.: An Aspectual Use Case Driven Approach. In: Pimentel, E., Brisaboa, N.R., Gómez, J. (eds.) VIII Jornadas de Ingeniería de Software y Bases de Datos (JISBD 2003), pp. 463–468. Alicante (2003)
4. Chitchyan, R., et al.: Survey of Analysis and Design Approaches, http://www.aosd-europe.net/deliverables/d11.pdf
5. Chung, L., Nixon, B.A., Yu, E., Mylopoulos, J.: Non-Functional Requirements in Software Engineering. Kluwer Academic Publishers, Dordrecht (2000)
6. Clarke, S., Baniassad, E.: Aspect-Oriented Analysis and Design: The Theme Approach. Addison-Wesley, Reading (2005)
7. The AspectJ Project, http://www.eclipse.org/aspectj/
8. International Telecommunication Union: Recommendation Z.150 (02/03), User Requirements Notation (URN) – Language Requirements and Framework, http://www.itu.int/rec/T-REC-Z.150/en
9. International Telecommunication Union: Recommendation Z.151 (11/08), User Requirements Notation (URN) – Language definition, http://www.itu.int/rec/T-REC-Z.151/en
10. Jacobson, I., Ng, P.-W.: Aspect-Oriented Software Development with Use Cases. Addison-Wesley, Reading (2005)
11. jUCMNav website, University of Ottawa, http://softwareengineering.ca/jucmnav
12. Lencastre, M., Araújo, J., Moreira, A., Castro, J.: Towards Aspectual Problem Frames: an Example. Expert Systems Journal 25(1), 74–86 (2008)
13. Mussbacher, G., Amyot, D.: Goal and Scenario Modeling, Analysis, and Transformation with jUCMNav. In: 31st International Conference on Software Engineering (ICSE 2009), Companion volume, pp. 431–432. IEEE, Los Alamitos (2009)
14. Mussbacher, G., Amyot, D.: Heterogeneous Pointcut Expressions. In: Early Aspects, Workshop of the 31st International Conference on Software Engineering (ICSE 2009), Vancouver (2009)
15. Mussbacher, G., Amyot, D.: On Modeling Interactions of Early Aspects with Goals. In: Early Aspects, Workshop of the 31st International Conference on Software Engineering (ICSE 2009), Vancouver (2009)
16. Mussbacher, G., Amyot, D., Weiss, M.: Visualizing Early Aspects with Use Case Maps. In: Rashid, A., Aksit, M. (eds.) Transactions on AOSD III. LNCS, vol. 4620, pp. 105–143. Springer, Heidelberg (2007)

17. Mussbacher, G., Amyot, D., Araújo, J., Moreira, A., Weiss, M.: Visualizing Aspect-Oriented Goal Models with AoGRL. In: Second International Workshop on Requirements Engineering Visualization (REV 2007). IEEE Computer Society Press, Los Alamitos (2007)
18. Mussbacher, G., Amyot, D., Weigert, T., Cottenier, T.: Feature Interactions in Aspect-Oriented Scenario Models. In: Reiff-Marganiec, S., Nakamura, M. (eds.) Feature Interactions in Software and Communication Systems X, pp. 75–90. IOS Press, Amsterdam (2009)
19. Mussbacher, G., Amyot, D., Whittle, J., Weiss, M.: Flexible and Expressive Composition Rules with Aspect-oriented Use Case Maps (AoUCM). In: Moreira, A., Grundy, J. (eds.) Early Aspects Workshop 2007 and EACSL 2007. LNCS, vol. 4765, pp. 19–38. Springer, Heidelberg (2007)
20. Mussbacher, G., Whittle, J., Amyot, D.: Semantic-Based Interaction Detection in Aspect-Oriented Scenarios. In: 17th International Requirements Engineering Conference (RE 2009). IEEE Computer Society Press, Los Alamitos (to be published, 2009)
21. Mussbacher, G.: Aspect-Oriented User Requirements Notation: Aspects in Goal and Scenario Models. In: Giese, H. (ed.) MoDELS Workshops 2007. LNCS, vol. 5002, pp. 305–316. Springer, Heidelberg (2008)
22. Mylopoulos, J., Castro, J., Kolp, M.: Tropos: A framework for requirements-driven software development. In: Brinkkemper, J., Solvberg, A. (eds.) Information Systems Engineering: State of the Art and Research Themes, pp. 261–273. Springer, London (2000)
23. Pourshahid, A., Mussbacher, G., Amyot, D., Weiss, M.: An Aspect-Oriented Framework for Business Process Improvement. In: Babin, G., Kropf, P., Weiss, M. (eds.) MCeTech 2009. LNBIP, vol. 26, pp. 290–305. Springer, Heidelberg (2009)
24. Rashid, A., Moreira, A., Araújo, J.: Modularisation and Composition of Aspectual Requirements. In: 2nd International Conference on Aspect Oriented Software Development (AOSD), pp. 11–20. ACM Press, New York (2003)
25. URN Virtual Library, http://www.usecasemaps.org/pub
26. van Lamsweerde, A., Letier, E., Darimont, R.: Managing Conflicts in Goal-Driven Requirements Engineering. IEEE Transactions on Software Engineering 24(11), 908–926 (1998)
27. Whittle, J., Jayaraman, P., Elkhodary, A., Moreira, A., Araújo, J.: MATA: A Unified Approach for Composing UML Aspect Models based on Graph Transformation. In: Transactions on Aspect-Oriented Software Development. LNCS. Springer, Heidelberg (to be published)
28. Yu, E.: Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. In: 3rd IEEE International Symposium on Requirements Engineering, pp. 226–235. IEEE Computer Society Press, Los Alamitos (1997)

A UML Profile for Goal-Oriented Modeling

Muhammad R. Abid, Daniel Amyot, Stéphane S. Somé,
and Gunter Mussbacher

SITE, University of Ottawa, 800 King Edward, Ottawa, ON, K1N 6N5, Canada
{mabid006,damyot,ssome,gunterm}@site.uottawa.ca

Abstract. The Unified Modeling Language (UML) does not fully address the needs of some important modeling domains, including goals and non-functional requirements (NFR). However, UML can be extended and tailored through the definition of profiles. In this paper, we propose a UML profile for the Goal-oriented Requirement Language (GRL), the goal/NFR notation of the User Requirements Notation (URN), recently standardized by ITU-T as Recommendation Z.151. Our profile is based on the abstract metamodel of GRL defined in accordance with ITU-T Recommendation Z.111 (meta-metamodel). This GRL metamodel has already been successfully tested and implemented in the jUCMNav Eclipse plug-in (a URN modeling tool). The profiling approach used in this paper adheres to the guidelines for UML profile design defined in ITU-T Recommendation Z.119. The resulting profile has been implemented in a UML 2 tool, namely IBM Rational Tau 4.0, and validated with case studies. Our experience and lessons learned are also discussed.

Keywords: Goal-oriented Requirement Language, Metamodel, Tau G2, UML Profile, User Requirements Notation.

1 Introduction

Goals are high-level objectives or concerns of a business, stakeholder, or system. They are often used to discover, select, evaluate, and justify requirements for a system. *Functional Requirements* (FR) define functions of the system under development, whereas *Non-Functional Requirements* (NFR) characterize system properties and qualities such as expected performance, robustness, usability and cost. Goals and NFRs capture essential aspects of systems, which have a significant impact throughout the development process. Goal models help to compare alternative requirements and solutions, facilitate trade-offs among conflicting concerns and constraints of different stakeholders, document rationales for design decisions (so we avoid revisiting previously considered and inferior solutions), and provide traceable justifications for lower-level requirements and design artifacts [4,19].

The Unified Modeling Language (UML) [16] is the most popular modeling language for developing software applications. However, many modelers are still unsatisfied with the role of UML in the area of goal and NFR modeling. With the importance of UML in the industry, this deficiency has now become an apparent weakness.

Modelers struggle to define how best to describe and structure goals. While some metamodels for goal modeling languages exist, such work has often been completed in isolation and has not been done in accordance with standards. Yet, there exists one mature metamodel for goal modeling, namely that of the Goal-oriented Requirement Language (GRL). GRL is combined with the Use Case Map (UCM) scenario notation to form the User Requirements Notation (URN) [11], recently standardized as ITU-T Recommendation Z.151 [12]. Standalone GRL-based modeling is useful on its own, and even more so when combined with UCMs. However, if one desires to integrate goal modeling with UML, then having a standardized and standalone goal metamodel may not be sufficient, and aligning it with the UML metamodel would help reduce existing communication and integration problems between goal modelers and UML modelers.

Although UML does not address explicitly the modeling of goals and non-functional requirements, there is a generic extension mechanism for tailoring UML to a particular domain, namely *UML profiling*. In this paper, we propose a UML profile for GRL. This profile satisfies the following properties, which are considerably important for successful goal-oriented modeling in a UML context.

1. *Integration with UML*: the ability to share information between the goal model elements and other UML elements.
2. *Diagram pollution avoidance*: preventing the mixing of different diagram constructs.
3. *Metamodel stability*: the maturity of the underlying goal metamodel.
4. *Implementability of the profiling mechanism*: how well the approach used for the creation of the profile is amenable to implementation and tool support.

A good integration with UML allows one to maintain traceability between goal models and UML models. The ability to avoid diagram pollution is useful to ensure the consistency of the goal diagrams created, and this is typically achieved by providing a dedicated diagram editor. A mature and stable metamodel contributes to the resulting profile's stability and validity. Finally, the implementability of a profile is important for integrating usable editors in existing UML modeling environments.

In our work, we satisfy the first property by virtue of UML profiling. Diagram pollution avoidance results from our choice of metamodeling implementation approach (*metamodel extension*, to be detailed in Sect. 3). Because our profile is based on a standardized metamodel (URN's, explored in Sect. 2), which has been already used as basis for GRL modeling tools [14], our work satisfies the stability property. The implementability of the profiling mechanism is demonstrated by a proof-of-concept implementation based on a commercial UML tool, to be described in Sect. 4.

Our main contributions are hence the definition of a UML profile for goal modeling rooted in the standard GRL metamodel and its implementation in IBM Rational (previously Telelogic) Tau 4.0 [8]. UML's metaclasses are mapped to GRL's metaclasses in accordance with standard guidelines provided in ITU-T Recommendation Z.119 [10]. We also discuss the typical usage of this profile with a small example where GRL is used standalone in a model, and then where GRL diagrams are combined with selected UML diagrams in a model. Related work is discussed in Sect. 5, followed by our conclusions.

2 Goal-Oriented Requirement Language (GRL)

This section introduces the GRL notation, main concepts, and metamodel.

2.1 Overview of the GRL Notation

GRL is a language that focuses primarily on goal modeling. A subset of URN and a graphical language, GRL's major asset is to provide ways to model and reason about non-functional requirements and other high-level goals. With GRL, the modeler is primarily concerned with exposing "why" certain choices for behavior and/or structure were introduced, leaving the "what" and the "how" to other languages such as UCM and UML. GRL integrates some of the best elements of two well-known goal-oriented modeling languages, i* [20] and the NFR framework [4]. Major benefits of GRL over other popular notations include the integration of GRL with a scenario notation, the support for qualitative and quantitative attributes, and a clear separation of GRL model elements from their graphical representation, enabling a scalable and consistent representation of multiple views/diagrams of the same goal model.

The graphical syntax of GRL (see Fig. 1) is based on the syntax of the i* language [20]. There are three main categories of concepts in GRL: actors, intentional elements, and links. A GRL goal graph is a connected graph of intentional

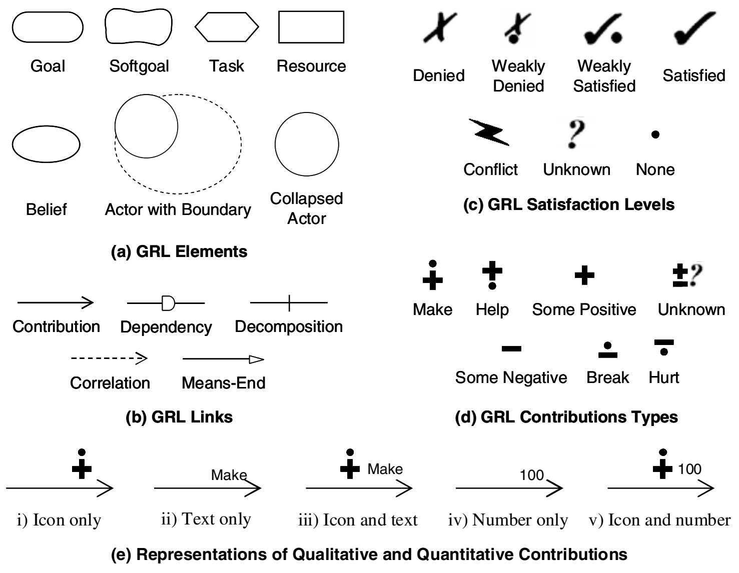


Fig. 1. Basic Elements of GRL Notation

elements that optionally reside within an actor boundary. An actor represents a stakeholder of the system or another system. Actors are holders of intentions; they are the active entities in the system or its environment who want goals to be achieved, tasks to be performed, resources to be available and softgoals to be satisfied. A goal graph shows the high-level business goals and non-functional requirements of interest to a stakeholder and the alternatives for achieving these high-level elements. A goal graph also documents beliefs (rationales) important to the stakeholder.

In addition to beliefs, *intentional elements* can be softgoals, goals, tasks, and resources. *Softgoals* differentiate themselves from *goals* in that there is no clear, objective measure of satisfaction for a softgoal whereas a goal is quantifiable. In general, softgoals are related more to non-functional requirements, whereas goals are related more to functional requirements. *Tasks* represent solutions to (or operationalizations of) goals or softgoals. In order to be achieved or completed, softgoals, goals, and tasks may require *resources* to be available.

Links (see Fig. 1.b) are used to connect elements in the goal model. *Decomposition links* allow an element to be decomposed into sub-elements. AND, IOR, as well as XOR decompositions are supported. XOR and IOR decomposition links may alternatively be displayed as *means-end* links. *Contribution links* indicate desired impacts of one element on another element. A contribution link can have a qualitative contribution type (see Fig. 1.d), or a quantitative contribution (integer value between -100 and 100, see Fig. 1.e). *Correlation links* are similar to contribution links, but describe side effects rather than desired impacts. Finally, *dependency links* model relationships between actors (one actor depending on another actor for something).

As an example, Fig. 2 shows a simple GRL model that illustrates the use of GRL constructs. This model captures various relationships between the concerns of customers, the bank and the merchant involved in some transaction payment. This is an artificial example used for testing our profile as it covers many combinations of links (including contributions with various weights), actors and intentional elements of different types in a single diagram. It will not be explained further here as it is not meant to reflect reality (for a GRL tutorials and literature, please refer to [3]).

2.2 GRL Metamodel

Figure 3 shows a graphical representation of the metamodel of the core GRL concepts, which constitute a part of the URN metamodel from Recommendation Z.151 [12]. These concepts represent the abstract grammar of the language, independently of the notation.

This metamodel formalizes all the GRL concepts and constructs introduced earlier. The GRL metamodel definition adheres to the guidelines of ITU-T Recommendation Z.111 [9] for metamodel-based definitions of ITU-T languages. This recommendation standardizes notations used to create abstract and concrete syntaxes of languages, using metagrammars or metamodels. Z.111 uses a subset of the Meta-Object Facility [15] to define a simple meta-metamodel targeting the definition of modeling languages.

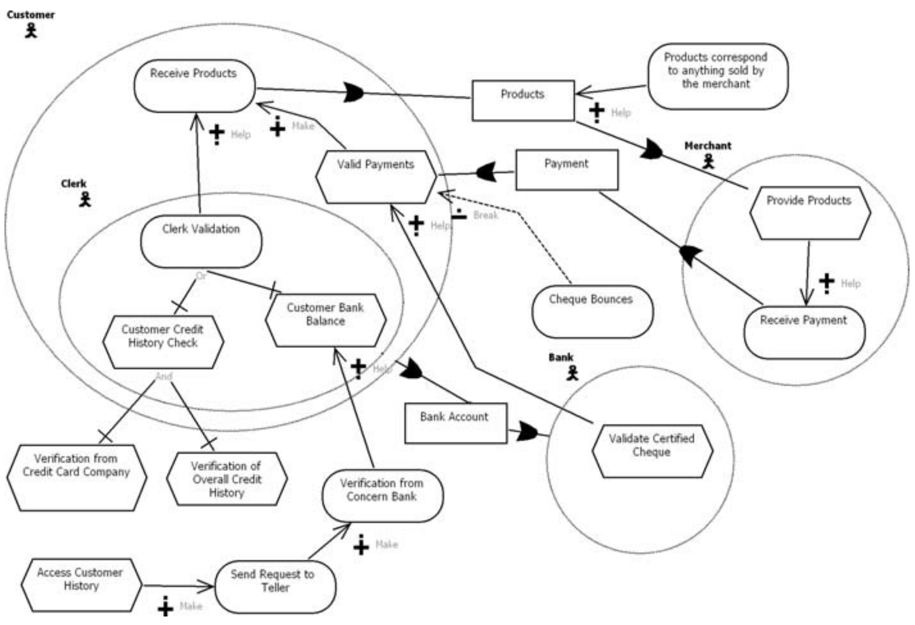


Fig. 2. GRL Model Example

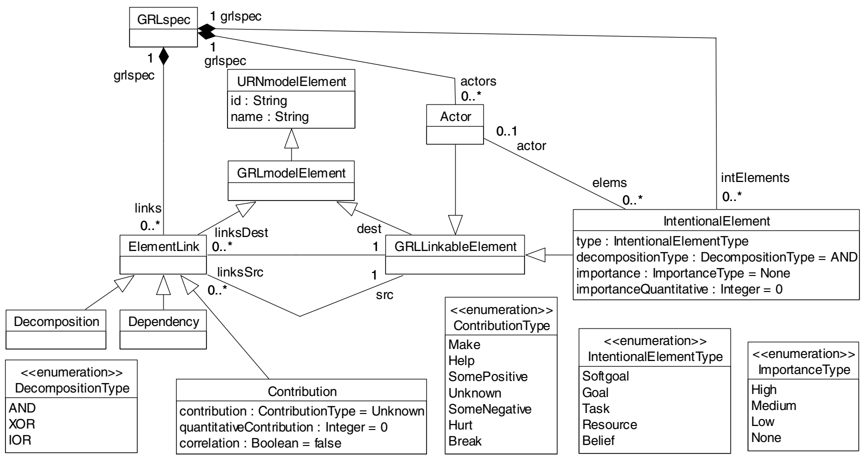

GRLspec
1 grlspec
1 grlspec
1 grlspec
URNmodelElement
id : String
name : String
GRLmodelElement
actors 0..*
Actor
0..1 actor
elems 0..*
intElements 0..*
IntentionalElement
type : IntentionalElementType
decompositionType : DecompositionType = AND
importance : ImportanceType = None
importanceQuantitative : Integer = 0
links 0..*
ElementLink
linksDest 0..*
dest 1
GRLLinkableElement
linksSrc 0..*
1 src
Decomposition
Dependency
Contribution
contribution : ContributionType = Unknown
quantitativeContribution : Integer = 0
correlation : Boolean = false
<<enumeration>> DecompositionType
AND
XOR
IOR
<<enumeration>> ContributionType
Make
Help
SomePositive
Unknown
SomeNegative
Hurt
Break
<<enumeration>> IntentionalElementType
Softgoal
Goal
Task
Resource
Belief
<<enumeration>> ImportanceType
High
Medium
Low
None

Fig. 3. GRL Metamodel (from Z.151)

3 UML Profile for GRL

The UML metamodel is not intended to be directly modifiable outside of the UML standardizing process. However, *profiling* enables one to customize UML to a particular domain. A new domain-specific language can hence be defined and integrated to the core UML by creating a profile that extends the UML metamodel. There already exist different UML profiles that have been developed by the Object Management Group [17], including the Systems Modeling Language (SysML), the Enterprise Distributed Object Computing (EDOC), and the Profile for Modeling and Analysis of Real-time and Embedded Systems (MARTE). This section discusses the creation of a profile for goal-oriented modeling based on GRL's concepts.

3.1 UML Profile Creation

Two main approaches are used for creating UML profiles: the *Stereotype Mechanism* (SM) and the *Metamodel Extension Mechanism* (MEM). The Stereotype Mechanism is a very common and straightforward method for creating a UML profile. This approach extends the basic UML elements and is supported by most UML tools. The different constructs used to define a profile are *Stereotype*, *Tagged value* and *Constraint* [16]. A stereotype is a metamodel construct defined as an extension of an existing UML metamodel element, in a manner similar to class extension using inheritance. A tagged value is similar to an attribute construct of a class in a UML metamodel. Tagged values are standard meta-attributes. Finally, constraints are restrictions on a metamodel required in a particular domain. Notice that existing UML metamodel constraints cannot be weakened in a profile. The definition of a profile using the SM method consists of:

1. Assigning a new name to an extended metaclass, which will be represented as a *stereotype* of UML;
2. Adding new attributes in the stereotype, which are called *tags*;
3. Adding new *constraints* to the stereotype.
4. Assigning a new appearance to the stereotype.

One limitation of the SM approach is that it does not prevent the mixing of domain-specific diagram constructs with predefined UML diagram constructs. This can result in *diagram pollution* and potentially hurt the understandability of models.

The Metamodel Extension Mechanism [13], which includes the functionalities of SM, is a less common method of creating a UML profile. It allows the extension of non-basic UML elements and imposes restrictions that ensure the sole use of domain-specific stereotypes in diagrams, thus avoiding diagram pollution. While more flexible, the MEM is a more complex approach of profiling and is supported by fewer tools than the SM. Given the desirable properties defined in Sect. 1, MEM is favored over SM in our work.

3.2 Overview of UML Profile for GRL

Our profile is based on the conventions defined in the ITU-T Recommendation Z.119 [10]. This recommendation defines specific rules and a template for ITU-T languages profiling. Because of space restrictions, only a summary of the profile is provided in this paper. The complete profile, documented in accordance to recommendation Z.119, is available in Abid's thesis [1]. Table 1 provides a list

Table 1. Summary of Stereotype Mapping Information

Stereotype	**Stereotyped UML Metaclass**
GRLspec	Model
GRLmodelElement	NamedElement
GRLLinkableElement	Class
Actor	Class
IntentionalElement	Class
IntentionalElementType	Enumeration
ImportanceType	Enumeration
ElementLink	Relationship
Contribution	Association
ContributionType	Enumeration
Dependency	Association
Decomposition	Association
DecompositionType	Enumeration

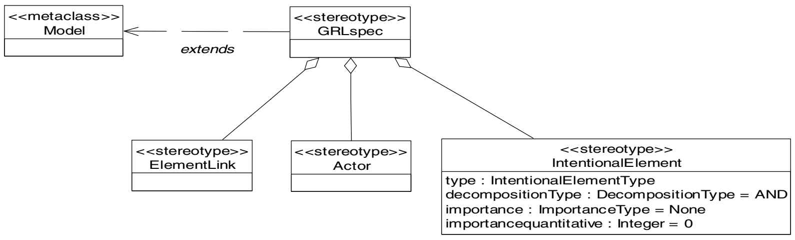

<<metaclass>>
Model
extends
<<stereotype>>
GRLspec
<<stereotype>>
ElementLink
<<stereotype>>
Actor
<<stereotype>>
IntentionalElement
type : IntentionalElementType
decompositionType : DecompositionType = AND
importance : ImportanceType = None
importancequantitative : Integer = 0

Fig. 4. GRL Specification

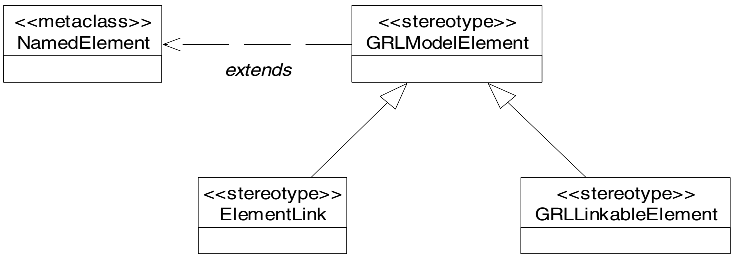
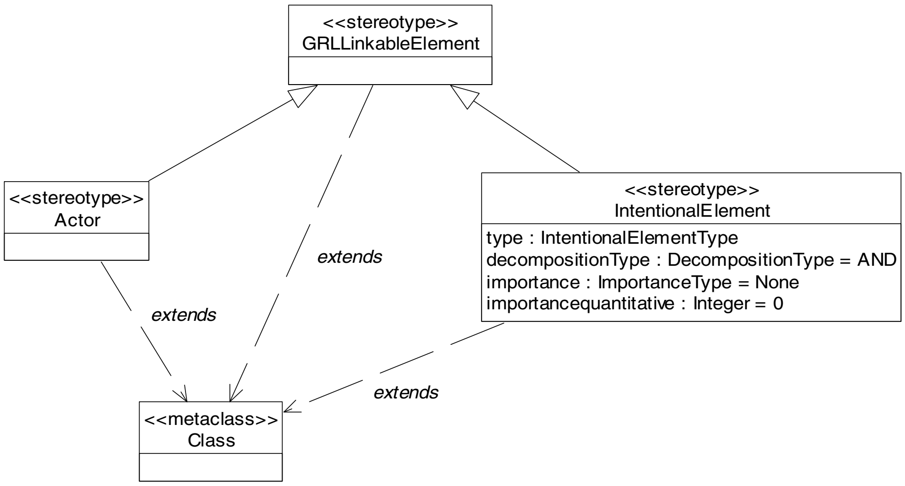


Fig. 5. GRL Model Element

of the stereotypes with the UML metaclass that each stereotype extends, while Figs. 4 to 7 graphically depict the UML profile with UML class diagrams.

Metaclass *GRLspec* from Fig. 3 maps to a stereotype of the UML metaclass *Model* as shown in Fig. 4. This is consistent with the fact that *GRLspec* is intended to serve as a container for GRL specifications, a role fulfilled by instances of *Model* in a UML specification.

GRLModelElement, which serves as superclass for the elements of a GRL model, extends the UML metaclass *NamedElement* (Fig. 5).



<<stereotype>>
GRLLinkableElement
<<stereotype>>
Actor
<<stereotype>>
IntentionalElement
type : IntentionalElementType
decompositionType : DecompositionType = AND
importance : ImportanceType = None
importancequantitative : Integer = 0
extends
extends
extends
<<metaclass>>
Class

Fig. 6. GRL Linkable Element

The stereotype *GRLLinkableElement* is intended for generalizing GRL *Actor* and GRL *IntentionalElement*. *GRLLinkableElement* extends the UML metaclass *Class* (see Fig. 6). Stereotypes *Actor* and *IntentionalElement* are also defined as extensions to metaclass *Class*. The UML metaclass *Class* describes a set of objects that share the same specifications of features, constraints and semantics. Its features are comprised of attributes and operations. Similarly, a GRL *Actor* has attributes and operations. Association *elems* between *Actor* and *IntentionalElement* is captured as a subset of association *nestedClassifier* owned by *Class* toward *Classifier*.

The stereotype *ElementLink* extends the UML metaclass *Relationship* (see Fig. 7) as its purpose is to show the intentional relationship among *GRLLinkableElements* which include stereotypes *Actor* and *IntentionalElement*.

The specializations of *ElementLink* (*Decomposition*, *Dependency* and *Contribution*) are defined from the UML metaclass *Association*. *Contribution* includes additional tagged values for the contribution type, quantitative value and correlation.

The enumeration types used in the profile (DecompositionType, ContributionType, IntentionalElementType and ImportanceType in Fig. 3) are simply kept as is.

3.3 Sample Details for a Profile Metaclass: IntentionalElement

For each metaclass, Abid's thesis [1] provides subsections with the details of the corresponding attributes, constraints, semantics, notation, and references to UML, as recommended in Z.119. This section illustrates such details for the stereotype *IntentionalElement*.

First, a correspondence between the attributes of the UML metamodel and those of GRL's is established. For example, *IntentionalElement* owns the

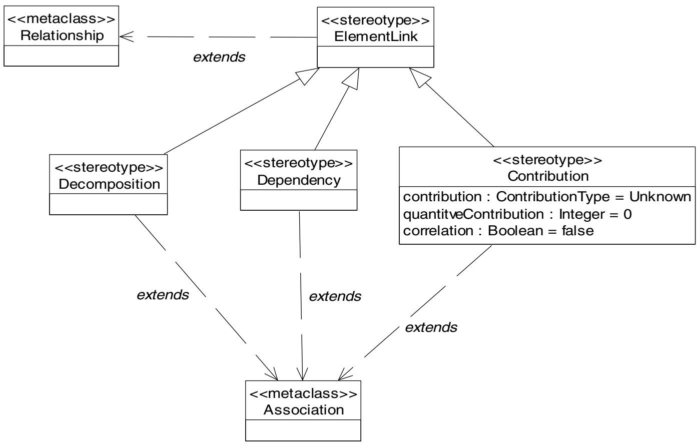


Fig. 7. Element Links

attributes listed in Table 2 and defined as tagged values. This table also includes the attributes from inherited metaclasses (*URNmodelElement*, via *GRLmodelElement*, see Fig. 3).

Constraints on the stereotypes are then described in a separate subsection of [1]. For instance:

- «IntentionalElement» <u>Class</u> has a tag importanceQuantitative whose <u>value</u> must be ≥ 0 and ≤ 100.
- Each «IntentionalElement» <u>Class</u> must have a unique <u>name</u>.

The semantics is also informally described in a separate subsection of [1]:

- The «IntentionalElement» <u>Class</u> has an association with GRL «Actor». It specifies the reasons for including particular behaviors, information and structural aspects in a system's requirements. There are different types of intentional elements corresponding to different types of behavior and information elements. These various types have different notations and can be linked to each other.
- The «IntentionalElement» <u>Class</u> has a tag importance that captures an actor's level of interest in the included intentional element. However, it is not mandatory that modelers use both the importance and importanceQuantitative tags. The selection depends on a modeler's requirements for the desired analysis type, either qualitative, quantitative, or mixed.

The notation is also described for each metaclass of the profile. For example:

- An «IntentionalElement» <u>Class</u> has different types as mentioned above in semantics. Each type has a separate notation: [1]

[1] The five first symbols in Fig. 1.a would be described here, but we do not repeat them because of space constraints.

Table 2. Attributes of *IntentionalElement*

Attribute	**Description**
name: String	Defines the name of the «IntentionalElement» Class.
id: String	Defines the identifier of the «IntentionalElement» Class.
type: Intentional-ElementType	This is an *enumeration* data type. It defines the different types of GRL «IntentionalElement»: *Softgoal, Goal, Task, Resource* and *Belief*.
decompositionType: DecompositionType	This is an *enumeration* data type. Its possible values are *AND, XOR* and *IOR*. Its default or initial value is *AND*. It defines the *decomposition* type when GRL «IntentionalElement» is the source of the decomposition link.
importance: ImportanceType	This is an *enumeration* data type. Its possible values are *High, Medium, Low,* and *None*. Its default value is *None*. It is used to evaluate the importance level of the intentional element to its owning actor when specified.
importance-Quantitave: Integer	Defines the evaluation of the quantitative importance of GRL «IntentionalElement» on its GRL «Actor». Its value ranges from 0 to 100, with 0 as default.

In the last subsection of [1], references to the relevant sections of the UML superstructure [16] are included:

– UML Superstructure: 7.3.7 Class (from Kernel).
– UML Superstructure: 7.3.33 NamedElement (from Kernel, Dependencies).

4 Implementation and Validation

In order to validate our UML profile for GRL, we have implemented it using IBM Rational Tau version 4.0 [8]. This modeling environment supports UML profiling. It allows extending the UML metamodel and, therefore, enables modelers to customize the UML metamodel for specific domains. Both the stereotype mechanism (SM) and the metamodel extension mechanism (MEM) are supported by the tool. However, some adaptation is required because the meta-metamodel supported by Tau (*TDDMetamodel*) differs from UML in a number of ways. For instance, the UML Enumeration and NamedElement metaclasses are absent or represented differently and the basic type String needed to be converted to Charstring. Hence, some customization of our profile was required during the implementation. Such differences in the meta-metamodel also leads to the generation of XMI files that may not be imported correctly by other UML tools, but tool interoperability was not a concern in our work so this was not tested.

Other UML tools supporting profiles were also considered: IBM Rational Software Architect was limited in its support of MEM [2], and Eclipse's Model Development Tools (MDT) was less mature than Tau at the time this work was done . However, MDT has evolved substantially since then, especially in terms of support for profiles and OCL-based validation of constraints [6].

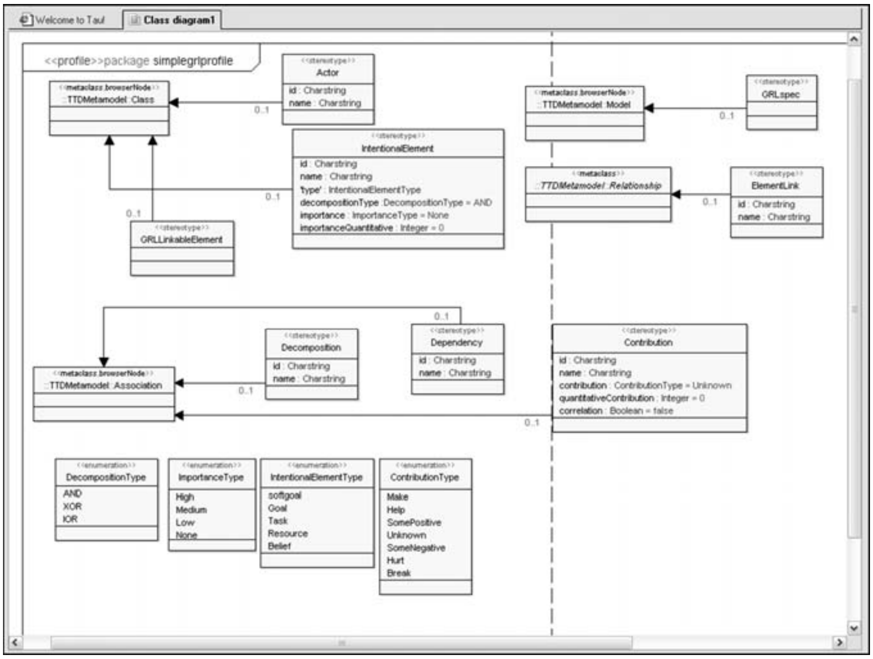

Welcome to Tau
Class diagram1
<<profile>>package simplegrlprofile
<<metaclass,browserNode>> ::TTDMetamodel::Class
<<stereotype>> Actor
id : Charstring
name : Charstring
0..1
<<metaclass,browserNode>> ::TTDMetamodel::Model
<<stereotype>> GRLspec
<<stereotype>> IntentionalElement
id : Charstring
name : Charstring
'type' : IntentionalElementType
decompositionType : DecompositionType = AND
importance : ImportanceType = None
importanceQuantitative : Integer = 0
<<metaclass>> ::TTDMetamodel::Relationship
<<stereotype>> ElementLink
<<stereotype>> GRLLinkableElement
<<metaclass,browserNode>> ::TTDMetamodel::Association
<<stereotype>> Decomposition
<<stereotype>> Dependency
<<stereotype>> Contribution
contribution : ContributionType = Unknown
quantitativeContribution : Integer = 0
correlation : Boolean = false
<<enumeration>> DecompositionType
AND
XOR
IOR
<<enumeration>> ImportanceType
High
Medium
Low
None
<<enumeration>> IntentionalElementType
softgoal
Goal
Task
Resource
Belief
<<enumeration>> ContributionType
Make
Help
SomePositive
Unknown
SomeNegative
Hurt
Break

Fig. 8. UML Profile for GRL with SM in Tau 4.0

Figure 8 gives an overview of the SM profile for GRL as defined in Tau. A drawback of the SM approach is that it does not allow the creation of a separate GRL editor. We must activate the GRL profile and then create a Class diagram. In the Class diagram editor, we create classes and manually associate an appropriate stereotype to each of these classes. A property view enables the setting of attribute values for the elements.

The MEM approach supported by Tau allows the generation of a specific GRL editor with a customized tool palette for GRL constructs. It is possible to "drag and drop" elements in the GRL editor to create a GRL model. The user is able to create a model based solely on GRL elements and avoid diagram pollution. Again, a property view can be used to provide values to attributes without visual representation. Furthermore, this approach allows to associate specific icons to the GRL elements. Figure 9 shows a view of a GRL editor created in Tau using the MEM approach.

Figure 10 gives a glimpse of a MEM-based implementation of the UML profile for GRL in Tau (which contains many more classes and connections to Tau's meta-metamodel than the SM metamodel). Such implementation is more complex as it requires one to connect the profile to predefined Tau stereotypes to allow/restrict visibility of elements in diagrams and the property view, and define icons, label positions, presentation, etc. In terms of semantics, this UML profile captures well the core GRL concepts described in Fig. 3, as these do not conflict with existing UML model elements. However, it is still incomplete as it does not

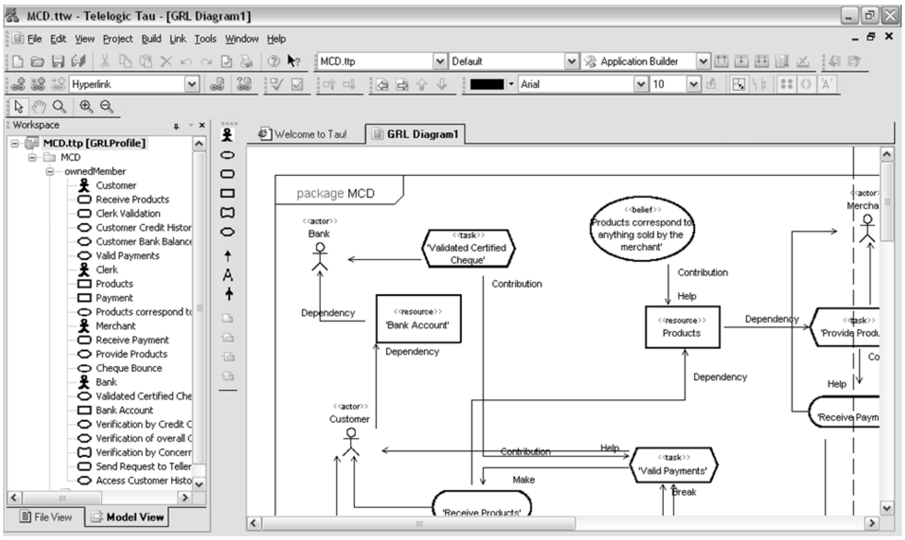


Fig. 9. Usage of UML Profile for GRL with MEM in Tau 4.0

yet support the notions of GRL strategy (which would be simple to add) and of connections to UCMs (on one hand harder to capture given their partial overlap with existing UML concepts, but on the other hand not absolutely required for goal modeling).

We evaluated our profile implementation by using the Tau generated editor on various examples, described in [1]. These experiments allowed us to verify the satisfaction of the properties listed in Sect. 1. In particular, GRL diagram elements are not intermixed with those from class diagrams (unlike the SM option). It is also possible to reuse GRL model elements in other diagrams: for example, one can drag a GRL actor from a GRL diagram to a UML use case diagram (to create an actor) or to a sequence diagram (to create a lifeline). Since these elements from different views refer to the same definition, changes to the name or another attribute of one of these elements are then automatically reflected in the other diagrams. Traceability links between GRL element and other UML elements can also be created and navigated. Such links are used to document design rationale and to explain decisions, so that weaker alternative designs are not constantly re-suggested. They also enable coverage assessments by answering questions such as "are all my goals sufficiently addressed by my UML design (goals without links)?" and "are there UML design elements that are spurious or that have no purpose (as they cannot be traced to any goal, directly or indirectly)?". This profile's implementation enables modelers to get answers to such important questions, which would be difficult to get otherwise.

We then compared the resulting diagrams with GRL diagrams obtained from jUCMNav [14], a dedicated URN modeling tool. A profile-based editor can hardly compete with a specialized editor from a usability perspective and in terms of

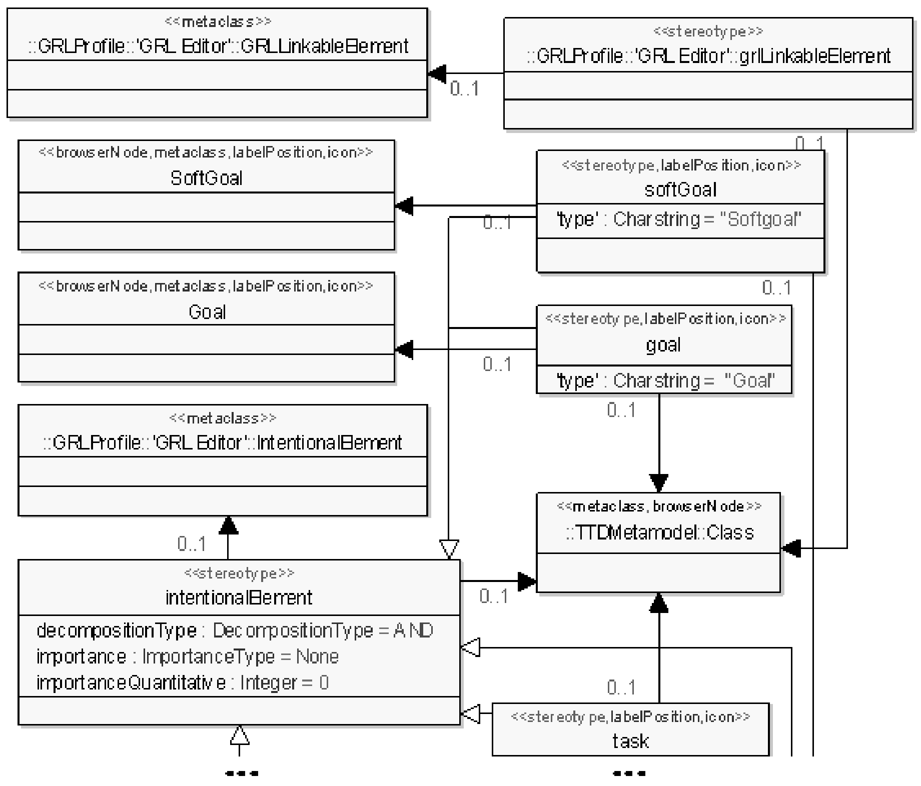

<<metaclass>>
::GRLProfile::'GRL Editor'::GRLLinkableElement
<<stereotype>>
::GRLProfile::'GRL Editor'::grlLinkableElement
0..1
<<browserNode,metaclass,labelPosition,icon>>
SoftGoal
<<stereotype,labelPosition,icon>>
softGoal
'type' : Charstring = "Softgoal"
<<browserNode,metaclass,labelPosition,icon>>
Goal
<<stereotype,labelPosition,icon>>
goal
'type' : Charstring = "Goal"
<<metaclass>>
::GRLProfile::'GRL Editor'::IntentionalElement
<<metaclass, browserNode>>
::TTDMetamodel::Class
<<stereotype>>
intentionalElement
decompositionType : DecompositionType = AND
importance : ImportanceType = None
importanceQuantitative : Integer = 0
<<stereotype,labelPosition,icon>>
task
...

Fig. 10. UML Profile for GRL with MEM in Tau 4.0 (Extract)

how good the diagrams look (compare jUCMNav's GRL diagram from Fig. 2 with Tau's in Fig. 9). Yet, it is actually possible to capture goal concepts, attributes, and relationships from within a UML tool (with much of the syntax and a palette supported), and integrate them to the rest of the UML model.

We were also able to identify some limitations of Tau that had some impact on the appearance of the resulting diagrams (some of which were already reported in [2]). Among others, adding icons to qualify links (e.g., qualitative GRL contributions) and visually embedding GRL intentional elements inside the boundaries of actors proved to be impossible with the version of Tau we used (see Fig. 9). The constraints defined in the profile, discussed in Sect. 3.3, cannot be easily supported as there is no constraint language such as UML's OCL (although constraint verification procedures could be defined in a language like C++ using the notion of Tau *agents*).

5 Related Work

Little research has addressed UML profiling for goal modeling. Supakkul and Chung [18] proposed a metamodel for NFR concepts that is integrated with the

UML metamodel through the use of a UML profile. The integration of UML and NFR notations occurs in a Use Case diagram. NFRs are represented as softgoals and associated with appropriate Use Case model elements. The authors implemented their work and illustrated it with a case study. A difference between this work and ours is that the former is restricted to NFRs. They also used an SM-based approach for the implementation, which does not prevent diagram (and concept) pollution.

A metamodel for enterprise goal modeling is proposed by Grangel *et al.* [7]. It distinguishes four different conceptual constructs: *Objective* (to represent targets that enterprises want to achieve), *Strategy* (to describe how the enterprise wants to achieve the proposed objectives), *Plan* (to represent the organization of the work at different hierarchal levels in order to accomplish the objectives and strategy) and *Variable* (to represent any factor that is able to influence the execution of the plans defined in the organization). A UML profile based on the metamodel has been implemented in IBM Rational Software Modeler Development Platform and in MagicDraw UML 12.0. Our goal metamodel appears more general and standard than the one in [7], which is specialized for enterprise goals. The profile implementation mechanism also appears to be an SM-based approach.

An approach for the integration of functional and non-functional requirements (NFR) is discussed by Cysneiros and Leite in [5]. The authors propose a junction point between the functional and the non-functional requirements development cycles named Language Extended Lexicon (LEL). The proposed approach is intended to cover all types of UML diagrams. For instance, a UML class diagram is integrated with related NFRs by having every root of each NFR graph refer to a LEL symbol and every class of the class diagram named using a LEL symbol. This approach is however not really defined as a UML profile. Similarly, van Lamsweerde presents an approach where goal modeling with KAOS is integrated with UML modeling, but not at the profile level [19].

6 Conclusion

In this paper, we presented a UML profile for goal-oriented modeling. Our research hypothesis is that UML can be profiled to support such modeling with a semantics rooted in a standard metamodel such as that of URN's Goal-oriented Requirement Language. A profile based on a mature and a stable metamodel that has been already used by editors and in analysis techniques is likely to represent a better alternative to the few existing solutions that currently exist and that were reviewed in the previous section. The definition of such profile should also be done according to standard recommendations such as Z.119. Z.119 provided us with useful guidance on the structure and the level of detail required for a good profile, but it was limited in guiding the selection of suitable metaclasses in UML corresponding to our GRL concepts. The implementation of the profile also enables models that can combine goal-oriented concepts with object-oriented concepts in a way that is comprehensible by the UML community at large.

We demonstrated that UML can indeed be profiled in such a way, and that it is possible to provide corresponding tool support through a commercial, profile-enabled UML environment (Tau in our case). An implementation approach based on UML's metamodel extension mechanism further enables the integration of goal modeling concepts with UML's in a tool while avoiding diagram pollution. Overall, this satisfies the desirable properties specified in the introduction.

Future extensions of this profile should include support for GRL strategies, which describe what-if scenarios to analyze GRL models and which are part of URN. The profile should eventually be extended to encompass URN entirely by also including Use Case Map scenario concepts. This would be in line with ITU-T's objective of having UML profiles for its formal languages (one such profile exists for SDL, namely Recommendation Z.109). Improvements to UML tools supporting profiles would also help cope with the complexity of supporting such modeling notations. The scalability, usability and acceptance by modelers of such a profile are also important research questions that deserve deeper studies.

Acknowledgments

This research was supported by the Natural Sciences and Engineering Research Council of Canada, through its programs of Discovery Grants and Postgraduate Scholarships.

References

1. Abid, M.R.: UML Profile for Goal-oriented Modelling. Master of Computer Science Thesis, University of Ottawa, Canada (2008)
2. Amyot, D., Farah, H., Roy, J.-F.: Evaluation of Development Tools for Domain-Specific Modeling Languages. In: Gotzhein, R., Reed, R. (eds.) SAM 2006. LNCS, vol. 4320, pp. 183–197. Springer, Heidelberg (2006)
3. Amyot, D., Mussbacher, G.: Development of Telecommunications Standards and Services with the User Requirements Notation. In: Workshop on ITU System Design Languages (2008), http://www.itu.int/dms_pub/itu-t/oth/06/18/T06180000010012PDFE.pdf
4. Chung, L., Nixon, B.A., Yu, E., Mylopoulos, J.: Non-Functional Requirements in Software Engineering. Kluwer Academic Publishers, Dordrecht (2000)
5. Cysneiros, L.M., Leite, J.C.S.P.: Using UML to Reflect Non-Functional Requirements. In: CASCON 2001: Proceedings of the 2001 conference of the Centre for Advanced Studies on Collaborative research. IBM Press (2001)
6. Eclipse.org: Model Development Tools (MDT), http://www.eclipse.org/modeling/mdt/
7. Grangel, R., Chalmeta, R., Campos, C., Sommar, R., Bourey, J.-P.: A Proposal for Goal Modelling Using a UML Profile. In: Enterprise Interoperability III, pp. 679–690. Springer, Heidelberg (2008)
8. IBM: Rational Tau, http://www-01.ibm.com/software/awdtools/tau/
9. International Telecommunications Union: Recommendation Z.111 (11/08), Notations to Define ITU-T Languages, http://www.itu.int/rec/T-REC-Z.111/en
10. International Telecommunications Union: Recommendation Z.119 (02/07), Guidelines for UML profile design, http://www.itu.int/rec/T-REC-Z.119/en

11. International Telecommunication Union: Recommendation Z.150 (02/03), User Requirements Notation (URN) – Language Requirements and Framework, http://www.itu.int/rec/T-REC-Z.150/en
12. International Telecommunication Union: Recommendation Z.151 (11/08), User Requirements Notation (URN) – Language definition, http://www.itu.int/rec/T-REC-Z.151/en
13. Jiang, Y., Shao, W., Zhang, L., Ma, Z., Meng, X., Ma, H.: On the Classification of UML's Meta Model Extension Mechanism. In: Baar, T., Strohmeier, A., Moreira, A.M.D., Mellor, S.J. (eds.) UML 2004. LNCS, vol. 3273, pp. 54–68. Springer, Heidelberg (2004)
14. jUCMNav website, University of Ottawa, http://softwareengineering.ca/jucmnav
15. Object Management Group (OMG): Meta Object Facility (MOF) Core Specification, Vers. 2.0, http://www.omg.org/docs/formal/06-01-01.pdf
16. Object Management Group (OMG): UML Superstructure Specification, Vers. 2.1.2, http://www.omg.org/docs/formal/07-11-02.pdf
17. Object Management Group (OMG): Catalog of UML Profile Specifications, http://www.omg.org/technology/documents/profile_catalog.htm
18. Supakkul, S., Chung, L.: A UML Profile for Goal-Oriented and Use Case-Driven Representation of NFRs and FRs. In: Third ACIS International Conference on Software Engineering, Research, Management and Applications (SERA 2005), pp. 112–121. IEEE Computer Society Press, Los Alamitos (2005)
19. van Lamsweerde, A.: Requirements engineering: From System Goals to UML Models to Software Specifications. John Wiley & Sons, Chichester (2009)
20. Yu, E.: Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. In: 3rd IEEE International Symposium on Requirements Engineering, pp. 226–235. IEEE Computer Society Press, Los Alamitos (1997)

Modelling a Debugger for an Imperative Voice Control Language

Andreas Blunk, Joachim Fischer, and Daniel A. Sadilek

Humboldt-Universität zu Berlin
Unter den Linden 6
D-10099 Berlin, Germany
{blunk,fischer,sadilek}@informatik.hu-berlin.de

Abstract. Creating debuggers for languages has always been a hard task. The main reason is that languages differ a lot, especially in the way programs are executed on underlying platforms. The emergence of metamodel-based technologies for defining languages simplified the creation of various language tools, e.g., creating editors from notation descriptions became common practice. Another, relatively recent, example is the metamodel-based description of execution semantics from which an interpreter can be derived. Such a semantics allows one to apply a model-based approach also to debugger development. In this paper, we demonstrate how a debugger can be modelled for an imperative voice control language. We show models of the debugging context, breakpoints, and stepping of voice control programs. These models are processed by a generic debugger.

1 Introduction

Debuggers are critical tools in software development. They are used by programmers to determine the cause of a program malfunction or simply to understand program execution behaviour. Programmers can follow the flow of execution and, at any desired point, suspend further execution and inspect the program's state. Execution may either be suspended manually or by setting breakpoints at well-defined program locations. A debugger then visualises the program's state. It presents all relevant information in the current context of execution, such as visible variable values and the program location.

Debuggers are well-known for general-purpose languages (GPLs). But, they can also be useful for executable domain-specific languages (DSLs). These languages are tailored to specific application domains. They allow developers to use specific concepts and notations to create programs of a corresponding domain with more concise and readable expressions than in GPLs.

In traditional language engineering, tools including debuggers are usually implemented by hand. But this can be too expensive for DSLs if they are used in small projects. It can also be a problem for bigger languages, e.g. UML or SDL, which are first specified in a standardisation process before tools are implemented by hand. Such manual implementation not only causes a gap between specification and tools but also delays tool availability.

A new language description technique that makes both language and tool development less expensive is metamodelling. It allows the description of the different aspects of a language with models, from which tools can be derived [1]. This is, for example, done for editors and interpreters, but there is currently no such technique for modelling debuggers.

The reason for this is that debuggers heavily depend on how language instances are executed and how runtime states can be extracted [2]. Execution semantics can either be described by a transformation to another language or by interpretation. In transformational semantics, debuggers depend on operating system capabilities and also on compilers and linkers that generate the symbol table and align target code. Debuggers for interpreted languages depend on language-dependent interpreter interfaces. Such dependencies make it hard to develop a modelling technique for debuggers and to implement a generic debugger.

What can already be implemented in a generic fashion is the graphical user interface part of a debugger because it is similar in many debuggers. The Eclipse Debugging Framework (EDF) [3] is such an implementation. It defines a set of Java-Interfaces, which concrete debuggers must implement. The EDF provides generic functionality on the basis of these interfaces. They forward user interactions to concrete implementations and they query them for debugging information that is displayed in the user-interface.

In this paper, we advance this state-of-the-art with a technique for *modelling debuggers*. It requires a metamodel-based description of the abstract syntax of a language and an operational semantics. Such language descriptions allow:

1. access to runtime states easily via model repositories, and
2. control of execution at the granularity of operational semantics steps.

Our approach is based on an EDF-based implementation of a *generic* debugger and descriptions for *specific* DSL debuggers. In contrast to *generated* tools, the *generic* debugger processes DSL-specific debugging descriptions and allows for domain-specific debugging of DSL programs. With our approach, the debugging of a DSL is described on the basis of its metamodel. It consists of various descriptions of debugging concepts: context information, program locations, breakpoints and step-operations. We demonstrate a description of these concepts with the sample DSL Voice Control Language (VCL).

In the following section, we present the foundations of our approach in more detail and we explain the language description of VCL in special example sections. Section 3 explains our approach for modelling debuggers. We demonstrate the modelling of a debugger for VCL in Sects. 4 and 5. The paper ends with a short conclusion and future work in Sects. 7 and 8.

2 Foundations

We use the Eclipse Modeling Framework (EMF) [4] as a metamodelling framework for MOF-based metamodels [5] and the operational semantics framework EProvide [6] for executing DSL programs according to an operational semantics description. Although we use EMF and EProvide, the approach is not limited to these tools. It may also be applied to other MOF-based metamodelling frameworks and other descriptions of operational semantics.

2.1 Metamodelling

A metamodel defines the abstract syntax of a language. It describes the structure of a set of possible language instances with object-oriented description means, e.g. classes, attributes, and associations. Language instances are models that contain instances of metamodel elements, e.g. objects, attribute values, and links.

Metamodelling frameworks such as EMF allow working with metamodels and models. They provide a model repository, editors and a programming environment, e.g. Java, that can be used to write programs on the basis of a metamodel.

Example 1 *An example DSL is the Voice Control Language (VCL). It is an imperative language that can be used to write programs for controller modules connected to a telephone. Besides concepts of imperative languages, e.g. variables and control structures, VCL also contains domain-specific concepts like* say text, perform action *and* listen for key press*. These concepts and their relations are defined by the metamodel, depicted in Fig. 1.*

Basically, VCL programs consist of reusable Modules *that contain sequences of* Actions*. There are domain-specific* Actions *like* SayText *and* Listen *but also ones that remind us of GPLs like* EnterModule*,* Decision *and* Assignment*.* ExpressionActions *can access* Variables *that save program state. They have to be declared local to a specific module or global to all modules. For simplicity reasons* Variables *are always of type Integer.*

A sample instance of the metamodel is depicted in Fig. 2. It is a gambling game, called probe game. The game can be played by calling a telephone that is connected to a corresponding controller. It randomly chooses between 0 and 1 and it tells you to make a guess. You either win the game with three successful probes or you lose it after three tries.

At the program level the global variables score *and* tries *are declared. Execution begins in the main module* probe*. It first outputs some information to the caller and then assigns initial values to the global variables. Next is a* while *action, which is executed if there are tries left. Execution then proceeds in the sub module* doProbe*. Modules can be compared to functions in GPLs. The module* doProbe *declares the local variables* in *and* result*. The first action listens for one of two possible inputs. It then assigns* in *to either 0 or 1. After the listen action, a random number between 0 and 1 is computed via an external system call. If the result matches the value of* in*, the caller gets a point. In either case the caller's tries are decremented. The module* doProbe *is left after execution of the last action has been completed. Execution continues after the* EnterModule *action in the main module* probe*. The section* startConfig *defines a number of initial inputs. They let a language developer test the program.*

The textual representation of VCL instances is defined with TEF [7]. It allows description of textual notations and derivation of textual editors automatically.

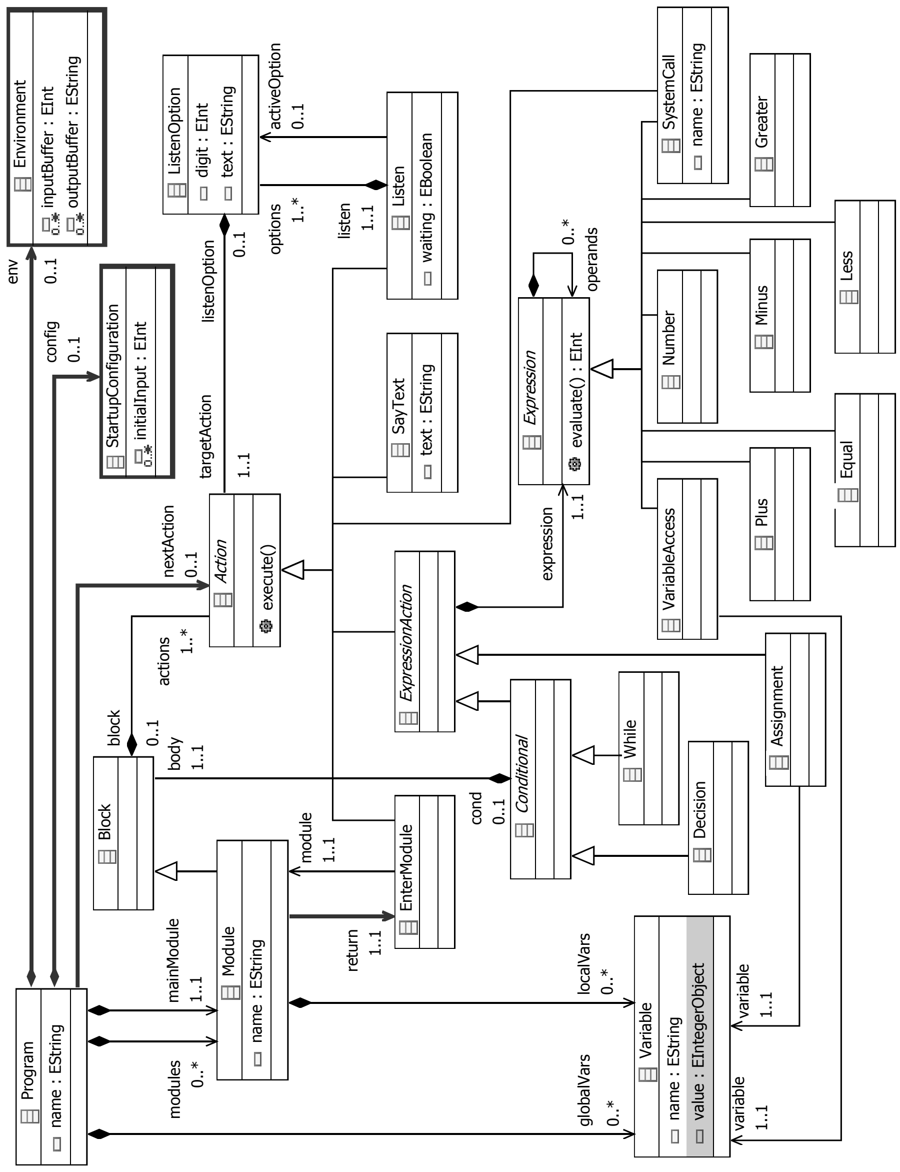


Fig. 1. VCL metamodel

2.2 Operational Semantics

An operational semantics defines the execution of language instances as a step-wise transition of a runtime state [8]. In a metamodel-based operational semantics, possible runtime states are modelled as part of a DSL metamodel and transitions are defined as model-to-model transformations. Such transformations can be defined with EProvide in one of various languages, e.g. Java, QVT, ASMs, Prolog or

```
program probeGame {
    decl score, tries;
    mainModule probe {
        say "probe on 0 or 1";
        score = 0;
        tries = 3;
        while tries > 0 {
            enter doProbe;
            if score = 3 {
                tries = 0;
                say "you win";
            }
        }
        if score < 3 {
            say "you lose";
        }
    }
```

```
    module doProbe {
        decl in, result;
        listen {
            0 : "probe 0" : in = 0;
            1 : "probe 1" : in = 1;
        }
        result = sys "random" (0, 1);
        if result = in {
            score = score + 1;
        }
        tries = tries - 1;
    }
    startConfig {
        input = [0, 1, 2];
    }
}
```

Fig. 2. Probe game written in VCL

Scheme. On the basis of such a definition, EProvide executes DSL instances stepwise. DSL developers can use the Eclipse launching dialog for specifying execution parameters, and they can control execution at the granularity of operational semantics steps. But up to now, EProvide does not support debugger features such as a variable view or more complex stepping.

Example 2 *To define the operational semantics of VCL programs, we extend the metamodel with a description of possible runtime states (emphasised elements in Fig. 1), e.g., the class* Environment*, which holds user inputs and program outputs, and the reference* Program.nextAction*, which plays the role of an instruction pointer.*

We define the transformation (step) and an initial state (reset) in Java (see Listing 1). It defines that in each step, one action is to be executed. Actions process inputs and outputs, which are contained in an Environment *object. Inputs are numeric keys pressed on a telephone key pad. They are consumed by* Listen *actions that define actions to be taken. Outputs are strings spoken to the caller. They are produced by* SayText *actions. All other actions define program state computations and conditional execution of actions.*

2.3 Model Transformations

In our approach, we also use model-to-model transformations in the transformation language QVT Relations [9]. A transformation is described on the basis of a source and a target metamodel. One specifies complex object patterns that, when found in a source model, result in matching another object pattern in a target model. If the target pattern does not exist, objects, links and values are created.

A transformation specification consists of a set of top-level relations that must hold in order for the transformation to be successful and ordinary relations that

```
public class JavaSemanticsProvider implements ISemanticsProvider {

  public void step(Resource model) {
    Program program = getProgram(model);
    ...
    Action action = program.getNextAction();
    if (action == null) return;
    if (action instanceof SayText) {
      program.getEnv().getOutputBuffer().add(((SayText) action ).getText ());
    }
    else if (action instanceof Listen) { ... }
    program.setNextAction (...);
  }

  public void reset (Resource model) {
    Program program = getProgram(model);
    Action firstAction = program.getActions().get (0);
    program.setNextAction( firstAction );
    program.getEnv(). getInputBuffer (). clear ();
    program.getEnv().getOutputBuffer(). clear ();
  }
}
```

Listing 1. Operational semantics description for VCL programs

are executed conditionally in *when* and *where* clauses. A relation needs to hold only when the relations in its *when* clause hold. A relation called in a *where* clause needs to hold only when the calling relation holds.

3 An Approach for Modelling Debuggers

The DSLs that are used in our approach are special in two ways. First, their runtime state is completely contained in a model. This makes it possible to describe a debugging representation of a DSL instance on the basis of its metamodel. Such a representation is defined by a model-to-model transformation of DSL instances to instances of a debugging metamodel, e.g. in QVT Relations. The debugging metamodel describes concepts for visualising threads, stack frames, variables and values; its instances are mapped to objects in EDF, which are displayed in the user-interface. Besides such state information, program locations are another part of a program's runtime state. Model objects that represent program locations are extracted by model queries and then highlighted in a concrete syntax.

The second characteristic is the step-wise execution of DSL instances. This makes the implementation of a generic debugger possible, which checks a program for active breakpoints to suspend further execution. Breakpoints are based on possible program locations. They can be installed for model objects that may represent program locations. Execution automatically suspends if an object that was marked as a breakpoint is included in a query for program locations.

Step-Operations, e.g., step-into, step-over, and step-return, are described similarly by model queries that extract model objects for target program locations. For such target locations, temporary breakpoints are installed and execution automatically suspends when one of those breakpoints is reached.

4 Debugging Context

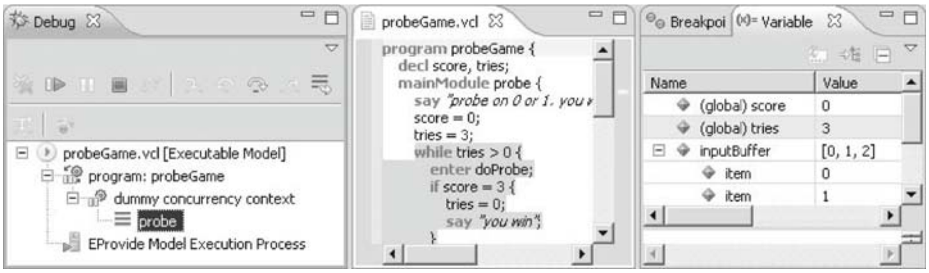
The presentation of context information is one of the major tasks of a debugger. Context information exists when the execution of a program is suspended. It tells a user where execution currently resides, how execution reached this point and what the values of visible variables are. Context information is derived from the runtime state of a program and displayed in different views in the user-interface of a debugger. The location where execution currently resides is referred to as the *current program location*. It is usually highlighted in an editor for a language. Information about how execution reached a program location includes:

1. concurrency information, e.g. information about threads, and
2. information about program parts that were activated during execution.

An example for activations are stack-frames that are created for function invocations. We refer to runtime elements that contain such activations as *activation frames*. Each concurrency context contains a sequence of activation frames that reflect the point in time where parts of a program were activated. By selecting an activation frame, information about variable structures is displayed as variables and values.

Example 3 *Figure 3 displays the context information of a VCL program. We can see that the current program location is a* While *action (editor in the middle), that the module probe has been entered to reach this location (left side) and that the variable score is allocated to the value 0 (right side).*

The context information that is displayed depends on the current selection of elements in a debugger's user-interface. An example is the selection of activation frames, which determines the presentation of visible variables and values. We define a *debugging context* to include runtime information of a suspended program


Debug
probeGame.vcl [Executable Model]
program: probeGame
dummy concurrency context
probe
EProvide Model Execution Process
probeGame.vcl
Breakpoi
Variable

```
program probeGame {
  decl score, tries;
  mainModule probe {
    say "probe on 0 or 1. you
    score = 0;
    tries = 3;
    while tries > 0 {
      enter doProbe;
      if score = 3 {
        tries = 0;
        say "you win";
      }
```

Name	Value
(global) score	0
(global) tries	3
inputBuffer	[0, 1, 2]
item	0
item	1

Fig. 3. VCL debugger

that is relevant to debugging, depending on the current selection of debugger elements. A debugging context includes the following types of information:

1. a set of concurrent execution contexts,
2. a sequence of currently activated program parts as activation frames,
3. a set of visible variables and their values, and
4. a program location.

Information types 1-3 are referred to as *debugging state information* and are not described in the same way as *program locations*. There can be many debugging contexts and many debugging states in a suspended program.

4.1 Debugging State

Debugging state information is represented as structured data in two different views of a debugger. Such data can be described by a metamodel as depicted in Fig. 4. The metamodel defines concepts for representing all possible debugging states and their relationships. The presentation of one of these debugging states depends on the current selection of an activation frame. The metamodel is thus referred to as *debugging states metamodel*.

All concepts have a textual representation that is defined by the attribute *labelText* in class *LabeledElement*. The root element of a debugging state is an *MProgramContext*. It contains information about concurrently executing program parts as *MConcurrencyContext*s. Each such context holds a sequence of activated program parts as *MActivationFrame*s. Activation frames contain visible variables and their values as *MVariable*s and *MValue*s. A variable may also contain other variables.

The generic debugger processes debugging state models and presents them in the user-interface. What has to be supplied is a mapping of possible runtime

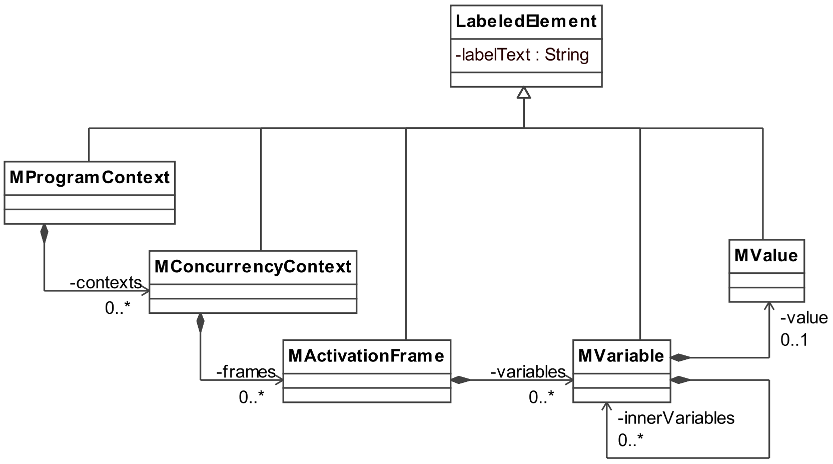


Fig. 4. Debugging states metamodel

states of DSL programs to instances of the debugging states metamodel. Such a mapping can be defined as a model-to-model transformation in QVT Relations. On the basis of a mapping the generic debugger computes a debugging state model and display its content.

Example 4 *For VCL, the mapping is defined in 115*[a] *lines of QVT Relations statements. Table 1 summarises the mapping. The table shows how VCL instances and their attributes are mapped to debugging states instances. We use the colon notation* o:Class *to indicate the presence or the creation of an object* o *for class* Class *and we use the punctuation notation* o.assocEnd *for referring to attributes or association ends.*

The first row indicates that each instance of Program *is mapped to an instance of* MProgram *plus an instance of* MConcurrencyContext. *VCL does not define concurrent execution of program parts. Therefore a mapping to a dummy* MConcurrencyContext *is necessary in order to map activation frames. The program's name* p.name *maps to the label of* MProgramContext mp.labelText. *The module that the next action is contained in* p.nextAction.module *maps to an activation frame in* cc.frames. *If the module was entered from another action and thus* p.nextAction.module.return *is not null, an activation frame is created in* cc.frames. *The mapping continues recursively for the entering action. After these explanations, the rest of the table should be comprehensible to the reader.*

An excerpt of the QVT Relations transformation is displayed in Listing 2. The top-level relation localVar *corresponds to the emphasised row in Table 1. It maps local VCL variables to debugging state variables. The relation needs to hold only if the relation* frame(m,af) *holds, i.e. there must be an activation frame* af *for the module* m. *The source object pattern checks for local variables and enforces corresponding* MVariable*s to exist. The transformation then continues in the where clause, which maps the variables values.*

With such a transformation, a debugging states model can be created and displayed by the generic debugger, e.g. the left and right side of the VCL debugger in Fig. 3.

[a] Line measurements in this paper do not include comments, empty lines and lines with ending braces.

4.2 Program Location

A program location is a highlighting of an element in the notation of a program that is somehow related to the current point of execution. There are different kinds of program locations. *Current program locations* exist for every concurrent execution context. They highlight a part of a program that will be executed when execution continues. Besides current program locations, debuggers usually also display *context-dependent program locations* for selected activation frames. These program locations highlight a part of a program that is currently being executed.

In GPLs, program locations are often displayed as highlighted statements in a textual concrete syntax. These statements are derived from some kind of instruction pointer. But metamodel-based DSLs are not necessarily textual and they do

Table 1. Mapping VCL instances to debugging states instances

VCL instance	**debugging states instance**
p:Program	mp:MProgram, cc:MConcurrencyContext
p.name	mp.labelText
	cc.labelText = ’dummy concurrency context’
p.nextAction.module	cc.frames
p.nextAction.module.return ...	cc.frames
p.env	cc.frames.variables
p.globalVars	cc.frames.variables
m:Module	af:MActivationFrame
m.name	af.labelText
m.localVars	*af.variables*
v:Variable	mv:MVariable
v.name	mv.labelText
v.value	val:MValue, val.labelText = v.value, mv.value = val
e:Environment	env:MVariable
e.inputBuffer	in:MVariable, env.innerVariables = in
e.inputBuffer.EInt	iv:MValue, iv.labelText = EInt, in.value = iv
e.outputBuffer	out:MVariable, env.innerVariables = out
e.outputBuffer.EString	ov:MValue, ov.labelText = EString, out.value = ov

not need to define explicit instruction pointers. Generally, a program location of such DSLs results from arbitrarily connected objects and their attribute values. Possible program locations are described by formulating an OCL [10] query that extracts model objects from the runtime state of a program. These model objects represent program locations and are highlighted in a notation of the program.

An example are Petri nets. The current program locations of a Petri net are determined by active transitions that are the result of a set of place objects and their markings. Program locations cannot be described by identifying static structures in a Petri net metamodel because there does not have to be an explicit reference to active transitions. Instead, a model query is necessary that extracts them.

Model queries for program locations are naturally described by OCL queries. But practical realisation requires a connection to an OCL editor, which we did not implement in our approach. Instead, program locations have to be described by implementing a Java interface that is defined by the generic debugger. In order to describe program locations in OCL, some additional code is necessary that evaluates OCL queries via MDT OCL[1] [11]. The generic debugger processes such descriptions when execution has suspended and informs appropriate editors that queried objects need to be highlighted.

The problem when highlighting current program locations is that there can be many such locations if there are multiple concurrent execution contexts. Context information is used to restrict these locations to one context-dependent

[1] MDT OCL is an Eclipse-based OCL implementation for EMF.

```
transformation RuntimeStateToDebuggingState(vclModel:vcl, dsModel:debuggingstate) {
  top relation localVar {
    n: String;
    checkonly domain vclModel m : vcl::Module {
      localVars = lv : vcl :: Variable {
        name = n
      }
    };
    enforce domain dsModel af : debuggingstate::MActivationFrame {
      variables = var : debuggingstate::MVariable {
        labelText = n
      }
    };
    when { frame(m,af); }
    where { lv.value.oclIsUndefined() or value(lv,var); }
  }
  ...
```

Listing 2. QVT transformation for mapping VCL programs to debugging state models

program location. The interface that has to be implemented defines the operations *getCurrentLocations* and *getLocationInActivationFrame* (see Listing 3). The first operation is purely used for breakpoint checking, which is explained in Sect. 5.1, and only the second operation is actually used for highlighting program locations on the basis of a selected activation frame.

Example 5 *In VCL, there is only one current program location, which is determined by the next action to be executed. This action is defined by the current allocation of* Program.nextAction*. Thus, the program location can be described by the following OCL query:* `Program.allInstances()->collect(p : Program | p.nextAction)->asSet()`*. It extracts all instances of* Program *and for the one program that exists, it retrieves the current allocation of* nextAction*. The complete Java implementation consists of around 30 lines of code.*

5 Execution Control

Basic execution control is already part of the generic interpreter EProvide. It allows to start, suspend, and terminate execution and to step forward and backward at the granularity of operational semantics steps. For a full-featured debugger, breakpoints and additional step-operations are necessary.

5.1 Breakpoints

Breakpoints are markings of possible program locations where execution should automatically be suspended. The generic debugger inspects the current program

```
public class VclSyntaxLocationProvider implements ISyntaxLocationProvider {

  public Collection <EObject> getCurrentLocations(Resource model) {
    // query model objects by using the Java API of MDT OCL
    // or by accessing the model with Java directly.
  }

  public EObject getLocationInActivationFrame(EObject dslFrame) {
    if (dslFrame instanceof Module) {
      Module module = (Module) dslFrame;
      Action nextAction = ((Program) module.eContainer()).getNextAction();
      // determines the currently executing action in the given module recursively.
      return getActionForModule(nextAction, module);
    }
    return null;
  }
  ...
```

Listing 3. Description of current program locations for VCL

locations for breakpoint markings after each operational semantics step and suspends or continues further execution. This way, DSL developers do not need to describe the reaching of breakpoints explicitly. What has to be supplied is a description of a marking function that checks whether a breakpoint marking for certain model objects should be allowed or not. The function has to be defined by implementing the Java interface *IBreakpointDescription*. It is used by the generic debugger when a user selects model objects in Eclipse (see Fig. 5). If the marking function evaluates to true, a special breakpoint activation entry is added to the context menu. Such a breakpoint description makes it possible to implement the breakpoint parts of the generic debugger in a completely generic way.

Example 6 *For VCL, breakpoints can be added only to* Actions. *The code in Listing 4 shows an implementation in Java.*

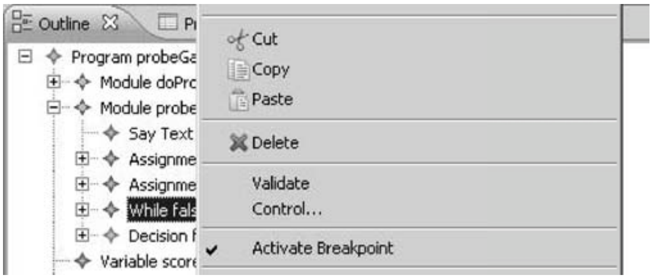


Fig. 5. Context menu for activating breakpoints

```
public class VclBreakpointDescription implements IBreakpointDescription {
  public boolean isBreakpointable(EObject object) {
    return object instanceof Action;
  }
}
```

Listing 4. Description of breakpoints for VCL

```
public class VclSyntaxLocationProvider implements ISyntaxLocationProvider {
   ...
  public Collection <EObject> getStepOverLocations(EObject curLocation) {
      Collection <EObject> locations = new HashSet<EObject>();
          if (curLocation instanceof EnterModule || curLocation instanceof Listen
                  || curLocation instanceof Conditional) {
              Action action = (Action) curLocation;
              locations .add(action . getAfterAction ());
              return locations ;
          }
      return null;
  }
  public EObject getStepReturnLocation(EObject dslFrame) {
      if (dslFrame instanceof Module) {
          Module module = (Module) dslFrame;
          if (module.getReturn() != null) {
              module.getReturn().getAfterAction ();
          }
      }
      return null;
  }
}
```

Listing 5. Description of step-operations for VCL

5.2 Step-Operations

Step-Operations allow to continue execution to a certain point. In an operational semantics step-operations result in the execution of several transformations until a certain state is reached. The state is determined by extracting target program locations from a program. At these locations, temporary breakpoints are installed and execution suspends again when one of these breakpoints is reached.

There are different kinds of step-operations. A step-over executes invocations of functions to completion. It depends on a program location, which is determined by a currently selected activation frame. On the basis of such a location, target program locations are extracted from the program. Generally, there are many target locations because execution may proceed at one of several program locations. Such a situation arises if continuation causes the execution of a conditional expression, for example an if-expression.

A step-return executes a currently entered function until completion, i.e. execution continues at the caller of the function. The target of a step-return depends on a selected activation frame. On the basis of such a frame, target program locations are extracted from the program.

The step-operation step-into is the default operational semantics step. Like program locations, step-operations are also described by implementing the Java interface *ISyntaxLocationProvider*.

Example 7 *For VCL, step-operations are described in Java as displayed in Listing 5. A step-over can be performed for the actions* EnterModule*,* Listen *and* Conditional*. Execution continues at the action that is located right after the current action at the same branching level. For example, in a* Decision *action it is the action that follows the* Decision *action.*

The target of a step-return is extracted from the selected activation frame, which is in the case of VCL a module. The generic debugger keeps track of source objects that activation frames are created from. Such a source object is provided as parameter dslFrame*. The target is the action that follows the* EnterModule *action, that caused the entering of the current module.*

6 Related Work

An approach for generating debuggers for grammar-based DSLs is the DSL Debugging Framework (DDF) [12]. The abstract syntax of a DSL has to be defined by a grammar and its execution semantics by a transformation to a general-purpose language (GPL) like Java. DSL debuggers are based on a mapping between DSL and GPL code and a GPL debugger. While a mapping describes the debugging concepts of a DSL, the actual debugging process works on the GPL level by using the GPL debugger. Mapping information is used to map DSL debugging commands to GPL debugger commands and GPL debugger results back to DSL results. The approach is limited to textual languages and it needs a GPL debugger for the target language.

Other approaches like ldb [13] and cdb [14] concentrate on generic debuggers for the programming language C. These debuggers can be re-used for varying target architectures, i.e., varying operating systems, compilers and machine architectures. They define a machine independent interface for common debugger actions, e.g. setting breakpoints or reading variable values. The interface has to be implemented for each target architecture. It encapsulates the technical and machine-dependent parts of a C debugger. The debugger itself is implemented on the basis of the machine-independent interface. This approach is also limited to grammar-based languages. Furthermore, execution semantics need to be defined by a special compiler (lcc) that automatically generates information for the debugger.

7 Conclusion

We presented a novel approach for modelling debuggers of metamodel-based DSLs that have an operational semantics definition[2]. The complete debugging description for VCL programs consists of around 160 lines of different descriptions in OCL, Java, and QVT. We are confident that such a description is a lot smaller and less expensive than a manually implemented debugger, although a direct comparison has not yet been conducted.

8 Future Work

We believe that our approach can also be applied to other metamodel-based languages, e.g. UML activities and the Object Constraint Language OCL. Future work could deal with the description of debuggers for such languages. Our experience with the sample language VCL shows that the availability of descriptions of runtime states and operational semantics is the main obstacle. But if there are such descriptions, debugging can be described with little effort.

Another area of interest is how the mapping for debugging states is described. Our experience shows that the structure of debugging states is, without any or with little structural changes, already part of a DSL metamodel. Consequently, another way to define the mapping could be to add inheritance relations between the classes of a DSL metamodel and the classes of the debugging states metamodel. Debugging state classes could declare associations as derived and DSL classes could specify how these associations are derived. This way, every DSL instance would also be a debugging state instance and could instantly be processed by the generic debugger. We believe that such a mapping would be easier to define and would execute faster than a transformation in QVT Relations.

References

1. Scheidgen, M.: Adopting Meta-modelling for ITU-T Languages: Language Tool Prototypes as a by-Product of Language Specications. In: Workshop on ITU System Design Languages (2008), `http://www.itu.int/dms_pub/itu-t/oth/06/18/T06180000010042PDFE.pdf`
2. Rosenberg, J.: How Debuggers Work - Algorithms, Data Structures, and Architecture. John Wiley & Sons, Chichester (1996)
3. Eclipse Foundation: Eclipse Debugging Framework, `http://help.eclipse.org/ganymede/index.jsp?topic=/org.eclipse.platform.doc.isv/guide/debug.htm`
4. Eclipse Foundation: Eclipse Modeling Framework (EMF), `http://www.eclipse.org/modeling/emf`

[2] The presented approach for describing DSL debuggers and the implementation of the generic debugger, which is called MODEF, are part of a diploma thesis [15]. The thesis includes more detailed information and more complete examples, but is unfortunately only available in German.

5. Object Management Group (OMG): Meta Object Facility (MOF) Core Specification, Vers. 2.0, http://www.omg.org/docs/formal/06-01-01.pdf
6. Sadilek, D., Wachsmuth, G.: EProvide: Prototyping Visual Interpreters and Debuggers for Domain-Specific Modelling Languages. In: Schieferdecker, I., Hartman, A. (eds.) ECMDA-FA 2008. LNCS, vol. 5095, pp. 63–78. Springer, Heidelberg (2008)
7. Humboldt-Universität zu Berlin: Textual Editing Framework (TEF), http://www2.informatik.hu-berlin.de/sam/meta-tools/tef
8. Plotkin, G.: A Structural Approach to Operational Semantics. Technical Report (DAIMI FN-19), University of Aarhus (1981)
9. Object Management Group (OMG): Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, http://www.omg.org/docs/formal/08-04-03.pdf
10. Object Management Group (OMG): Object Constraint Language (OCL) Version 2.0, http://www.omg.org/docs/formal/06-05-01.pdf
11. Damus, C.W.: Implementing Model Integrity in EMF with MDT OCL, http://www.eclipse.org/articles/article.php?file=Article-EMF-Codegen-with-OCL/index.html
12. Wu, H., Gray, J., Mernik, M.: Grammar-Driven Generation of Domain-Specific Language Debuggers. Softw. Pract. Exper. 38(10), 1073–1103 (2008)
13. Ramsey, N.: A Retargetable Debugger. Ph.D. Thesis, Princeton University, Princeton (1993)
14. Hanson, D., Raghavachari, M.: A Machine-Independent Debugger. Softw. Pract. Exper. 26(11), 1277–1299 (1996)
15. Blunk, A.: MODEF – Ein generisches Debugging-Framework für domänenspezifische Sprachen mit metamodellbasierter Sprachdefinition auf der Basis von Eclipse, EMF und EProvide. Diploma Thesis, Humboldt-Universität zu Berlin (2009)

The ARC Programming Model

Kevin Marth and Shangping Ren

Illinois Institute of Technology, Chicago, IL 60616, USA
{martkev,ren}@iit.edu

Abstract. The ARC (Agent, Role, Coordination) programming model is an evolutionary synthesis of established software technology that helps model-driven pervasive software applications to effectively utilize the parallelism on multicore processors. Computational entities in the ARC programming model are composed of role-based agents, exposing natural opportunities for inter-entity and intra-entity parallelism and facilitating collaboration-based designs. The coordination required by collaboration-based designs is separated from other computation and enacted via coordination agents upon coordinated role-based agents. The implementation of the ARC programming model is responsible for mapping relatively abstract model-level parallelism to the target platform and exploiting the available processor-level parallelism.

1 Introduction

Computation is being influenced by the convergence of two technological trends: parallelism and pervasiveness. The trend toward parallelism is apparent in the direction of commodity microprocessors, where the emphasis is now on increasing the number of processors (cores) per chip. The new interpretation of Moore's Law states that the number of cores per chip will likely double every two to three years. The trend toward pervasiveness implies that open and distributed applications will continue to proliferate. The question of how to program pervasive applications to execute on parallel hardware is therefore an important one.

While the shift to multicore processors represents a revolution in computer hardware, the software response will necessarily be more evolutionary. Ideally, multicore processors could be exploited through synthesis and extension of established programming models and software technology. We believe that effective exploitation of multicore processors is facilitated by a programming model in which parallelism is intuitive, manifest, and relatively abstract. The desire for a higher level of abstraction aligns with approaches to model-driven engineering (MDE) [1] such as the model-driven architecture (MDA) [2]. In the MDA, the specification and design of a software application is separated from the specification of the platform technology used by the application. The basis of the MDA is a platform-independent model (PIM) of the system of interest and the (automated or manual) translation from a PIM to a platform-specific model (PSM). The effective exploitation of available hardware parallelism is then a concern of the toolchain that generates a deployable executable from the PIM and PSM.

A PIM may be specified using SDL-2000 [3] exclusively or using the Unified Modeling Language (UML) [4] combined with SDL-2000 [5]. In either case, SDL-2000 defines the executable semantics of the PIM. SDL-2000 embodies decades of experience in the development of the distributed systems prevalent in pervasive computing and also enables the specification of parallel computational agents. Conceptually, each parallel agent has a dedicated thread of control, although the PSM implementation may be specialized to the actual parallelism available on a particular processor provided the conceptual model is not violated. Exploiting massive multicore parallelism in an SDL-2000 MDE application requires the PIM to be effectively decomposed into parallel agents. Opportunities for parallelism should be exposed while retaining a convenient and intuitive programming model. It has been argued that if concurrency was intrinsically difficult, humans would not function well in a physical world filled with concurrent activity. Of course, humans and other entities, both animate and inanimate, often successfully play several roles concurrently. Further, the roles played by an entity that are of interest to observers determine the points-of-view from which the entity is considered. Consequently, the role concept has significantly influenced software development. We believe that parallel role-based agents offer a convenient and intuitive programming model for structuring an MDE application specified as an SDL-2000 PIM. The executable PSM generated for an SDL-2000 PIM exploits parallel role-based agents to efficiently utilize multicore parallelism while shielding application developers from the complexities of low-level synchronization.

A programming model with role-based agents also enables collaboration-based designs. A collaboration is a collection of roles played by participating entities that cooperate to achieve an application feature. The cooperation among roles within collaboration-based designs is a particular concern in the model of computation provided by SDL-2000. Parallelism among agents implies the need for coordination, but SDL-2000 does not enable the coordination of groups of agents to be specified in a convenient and modular fashion. Communication among agents may be asynchronous and may be subject to arbitrary delay, which implies that the order in which messages are received by agents may be nondeterministic. If an application requires an agent to process a set of messages received on multiple channels in a deterministic order, the application must specify such determinism within the behavior of the agent. Similarly, if an application requires that a set of messages be processed by a set of agents in a deterministic order, the application must specify message and behavior protocols that will result in the desired determinism. When such protocols are explicitly specified for individual agent behaviors, the protocols result in cross-cutting concerns that become entangled within agent behaviors.

The ARC (Agent, Role, Coordination) programming model helps pervasive software applications effectively utilize the parallelism on multicore processors. The ARC model uses an extended dialect of SDL-2000 (SDL-ARC) and is a novel but evolutionary synthesis of established software technologies, including role-based modeling, split objects, and aspect-oriented programming (AOP). The ARC model has several distinguishing features.

- Computational entities are composed of role-based agents, exposing inter-entity and intra-entity parallelism and enabling collaboration-based designs.
- The coordination required by collaboration-based designs is separated from other computation and enacted via coordination agents upon coordinated role-based agents.
- A program in the ARC model is a PIM, and the associated PSM implementation maps the explicit and relatively abstract model-level parallelism to the target platform, exploiting the available processor-level parallelism.

This paper examines the distinguishing features of the ARC programming model. Section 2 examines the structure of computational entities and the role-based agents that compose them. Section 3 examines the coordination required by collaboration-based designs. Section 4 surveys the realization of the ARC model. Section 5 discusses related work. Section 6 concludes the paper.

2 Agents, Entities, and Roles in the ARC Model

The ARC model is a conservative synthesis and extension of established software technology intended to help exploit multicore processors while preserving the accepted and familiar semantics of the contributing technology. This section introduces computational entities composed of SDL-2000 agents and organized on the basis of a synthesis of role-oriented modeling and split objects. The roles played by an entity are often relatively independent, which implies that the roles may effectively execute in parallel, subject only to data dependencies within the entity. This simple observation is fundamental to the design of the ARC model.

2.1 Agents

The agent is the fundamental unit of specification in SDL-2000. The specification of an agent may include three optional sections: local attributes (parameters, variables, procedures), behavior, and internal structure. The behavior of an agent is specified via a communicating Extended Finite State Machine (EFSM) [6]. The concurrency within an agent with internal structure is specified by defining the agent as either a `block` agent or a `process` agent. In a block agent, the behavior of the agent and the behaviors of all internal agents execute concurrently within dedicated conceptual threads of control. In a process agent, the behavior of the agent and the behaviors of all internal agents alternate execution within a single conceptual thread of control. An SDL-2000 system is a special block agent with the capability to communicate with the environment. The system agent is typically decomposed into block agents that represent subsystems, which may be further decomposed as appropriate. Ultimately, several process agents are typically required to interact, and it is at this level that the ARC programming

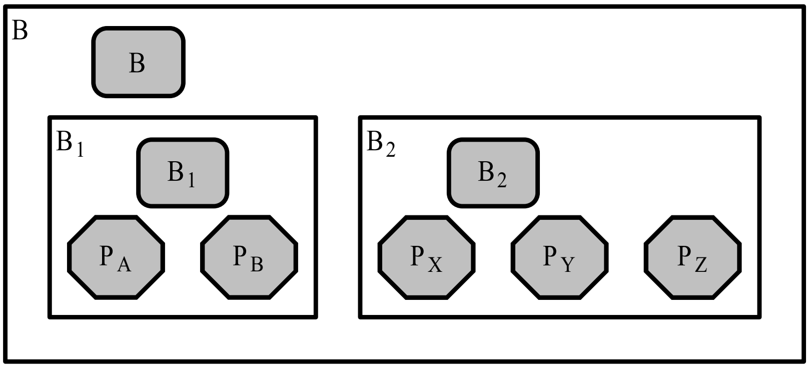


Fig. 1. Block and Process Agents in SDL-2000

model is focused. For example, Fig. 1 illustrates a block agent B with two internal block agents, B_1 and B_2. Agent B_1 is composed of process agents P_A and P_B. Agent B_2 is composed of process agents P_X, P_Y, and P_Z. There are eight conceptual concurrent threads of control, including the EFSM behaviors of the five process agents (denoted by octagons) and the EFSM behaviors of the three block agents (denoted by rounded rectangles).

The advent of multicore processors has triggered renewed interest in the Actor model [7], since the Actor model seems well-suited to the expected computational trends. A natural mapping exists between SDL-2000 agents and actors. SDL-2000 agents are active, autonomous, and encapsulated. Encapsulation ensures that interaction with an agent is accomplished strictly through signal exchange. Since the internal state of an agent is protected from direct external access, autonomy is ensured. Each agent has a unique, invariant identity and its own input queue where signals are received and buffered. An agent processes its signals serially and to completion. Thus, the complexities and hazards of concurrent access to the encapsulated state of an agent are not an issue in SDL-2000, which is a major advantage over multicore programming models in which explicitly synchronized access to shared memory predominates. The concurrency that exists both within and between SDL-2000 block agents is manifest and relatively abstract and safe. The combination of asynchronous and synchronous communication also enables an effective and intuitive parallel programming model. All agents with a dedicated conceptual thread of control execute asynchronously, and synchronous communication is used only when dictated by control or data dependencies among agents.

2.2 Entities and Roles

As multicore processors become massively parallel, it is important to maximize the parallelism among SDL-2000 agents. A software system is typically composed of a relatively small number of block agents that represent the structural hierarchy of subsystems within the application. The additional parallelism required to exploit massively multicore processors must be exposed at the level of the process agents that interact within subsystems. However, the required increase in parallelism must be balanced with the need for an intuitive and convenient programming model at the level of process agents. We believe that such a model can be synthesized from two established models that are based on the notion of software viewpoints: role-oriented modeling and split objects. In role-oriented modeling [8] the features of an object are classified as either intrinsic or extrinsic. The intrinsic features of an object are allocated directly to the object, while the extrinsic features are allocated to the various roles played by the object in order to collaborate with other objects in an application. The following properties of roles are commonly accepted.

- Abstractivity: Role generalization and inheritance hierarchies may exist.
- Aggregation/Composition: Roles can be composed of other roles.
- Dependency: A role cannot exist without an object. The methods of a role may be defined in terms of the methods of the object, but not vice versa.
- Dynamicity: An object may add and remove roles during its lifetime.
- Identity: An object and its roles constitute a single entity with one identity.
- Multiplicity: An object may play several roles simultaneously, including multiple instances of the same role.
- Visibility: Access to an object is restricted to the methods of a role, excluding methods of other roles.

In the split object model [9], a split object is a collection of parts that share a common identity. A split object denotes a single entity, and each part of a split object denotes a viewpoint of the entity. A natural synthesis of role-oriented modeling and the split object model results when the viewpoints of an entity are equated with roles, effectively partitioning an entity based on the roles it plays.

The synthesis of role-oriented modeling and split objects is realized in the ARC model by structuring computation in terms of entities composed of `basic` agents and `role` agents. Both basic agents and role agents are SDL-2000 process agents. A role agent is dependent on its parent agent within an entity. The parent of a role agent is either a basic agent, or in the case of role aggregation, another role agent. A basic agent may be created by an arbitrary agent, and no dependent parent relationship is implied. An entity is composed of one basic agent that represents the intrinsic part of the entity and zero or more role agents that represent the extrinsic parts of the entity. If the basic agent and role agents that compose an entity are defined in the context of a block agent, each part of the entity has a dedicated conceptual thread of control. Thus, the intrinsic and extrinsic parts of the entity may all execute in parallel, subject only to the data dependencies within the entity. The treatment of entities in the ARC model is descriptive and not prescriptive. An entity is a semantic concept and not a syntactic construct, and parent relationships within an entity are not enforced.

An example of an entity in the ARC model is illustrated in Fig. 2, where basic agents are denoted by squares, role agents are denoted by circles, and solid arrows indicate parent relationships among basic agents and role agents. The entity Joe has four agent parts. The intrinsic part of Joe defines features that are invariant across roles, e.g. the name of a person, and is implemented by an instance of the Person basic agent (P_J). The example assumes that Joe is a student who also works two jobs. Thus, an instance of the Student role agent (S_J) and two instances of the Employee role agent (E_{J1},E_{J2}) implement the extrinsic parts of Joe. Role agents define features that are specific to a role or viewpoint of an entity, e.g. the phone number of a person. The role property of dynamicity is apparent in the ability to dynamically create or delete role agents that implement extrinsic parts of an entity. The role property of multiplicity is apparent in the simultaneous existence of multiple roles for Joe, including two Employee role agent parts of Joe. To implement the identity role property, basic agents and role agents have an input signal named entity? [1]. The result of the entity? input for a basic agent is its own identity (self), and the entity? input for a role agent forwards entity? to its parent. Consequently, the identity of an entity is the identity of its intrinsic basic agent, and the identity returned by an entity? signal sent to any part of Joe is P_J.

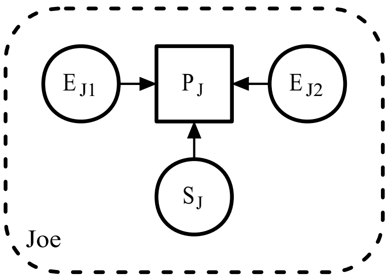


Fig. 2. Person Entity with Employee and Student Roles in the ARC Model

The SDL-ARC agents used to implement the entity Joe are specified in Fig. 3. The Main agent illustrates the role properties of dynamicity and multiplicity in its use of the Person basic agent and the Employee and Student role agents to create the entity Joe with the four agent parts illustrated in Fig. 2. The dependency role property is apparent in that input signals in a role agent, e.g. name, may be delegated to the parent agent, but input signals in a parent agent may not be delegated to role agent children. The Main agent also illustrates the role property of restricted visibility based on the viewpoint of a role, as each call to the phone procedure will assign a different phone number string. The abstractivity role property is based on agent type inheritance in SDL-2000. Both basic agents and role agents in the ARC model can use agent type inheritance to redefine agent features. The aggregation/composition role property (a role with a subrole) will be illustrated in the following section.

[1] The lexical rules for names in SDL-ARC allow a terminating ?, !, or $.

```
system PersonEntity; block B; syntype Agent = Pid endsyntype;
signal
  entity?(Agent), entity(Agent), newEmployee(Agent), newStudent(Agent),
  asEmployee(Charstring), asStudent(Charstring),
  name?(Agent), name(Charstring), phone?(Agent), phone(Charstring);

process Person (0,255) (name, phone Charstring);
signalset entity?, name?, phone?, asEmployee, asStudent;
dcl caller Agent; dcl phone1 Charstring;
start; nextstate main; state main;
  input entity?(caller); output entity(self) to caller; nextstate -;
  input name?  (caller); output name  (name) to caller; nextstate -;
  input phone? (caller); output phone(phone) to caller; nextstate -;
  input asEmployee(phone1); create Employee(phone1);
    output newEmployee(offspring) to sender; nextstate -;
  input asStudent( phone1); create Student( phone1);
    output newStudent( offspring) to sender; nextstate -;
endprocess;

process Employee (0,255) (phone Charstring);
signalset entity?, name?, phone?; dcl caller Agent;
start; nextstate main; state main;
  input entity?(caller); output entity?(caller) to parent; nextstate -;
  input name?  (caller); output name?  (caller) to parent; nextstate -;
  input phone? (caller); output phone   (phone) to caller; nextstate -;
endprocess;

process Student (0,255) (phone Charstring); /* ... */ endprocess;
/* Student EFSM identical to Employee EFSM in this example */

process Main (1,1);
signalset newEmployee, newStudent, name, phone, entity;
dcl E1,E2,E3,Joe,JoeE1,JoeE2,JoeS Agent; dcl S1,S2,S3 Charstring;

procedure entity$(agent Agent, in/out entity Agent);
start; output entity?(self) to agent; nextstate wait;
state wait; input entity(entity); return; save *; endprocedure;

procedure name$(agent Agent, in/out name Charstring);
start; output name?(self) to agent; nextstate wait;
state wait; input name(name); return; save *; endprocedure;

procedure phone$(agent Agent, in/out phone Charstring);
start; output phone?(self) to agent; nextstate wait;
state wait; input phone(phone); return; save *; endprocedure;

procedure asEmployee(agent Agent, phone Charstring, in/out role Agent);
start; output asEmployee(phone) to agent; nextstate wait;
state wait; input newEmployee(role); return; save *; endprocedure;

procedure asStudent(agent Agent, phone Charstring, in/out role Agent);
start; output asStudent(phone) to agent; nextstate wait;
state wait; input newStudent(role); return; save *; endprocedure;

start; task { create Person('Joe', '555-0000'); Joe := offspring; };
  call asEmployee(Joe, '555-1111', JoeE1); call entity$(JoeE1, E1);
  call asEmployee(Joe, '555-2222', JoeE2); call entity$(JoeE2, E2);
  call asStudent (Joe, '555-3333', JoeS);  call entity$(JoeS , E3);
  call assert(E1 = Joe and E2 = Joe and E3 = Joe);
  call phone$(JoeE1, S1); call phone$(JoeE2, S2); call phone$(JoeS, S3);
  call assert(S1 = '555-1111' and S2 = '555-2222' and S3 = '555-3333');
  call name$ (JoeE1, S1); call name$ (JoeE2, S2); call name$ (JoeS, S3);
  call assert(S1 = S2 and S2 = S3 and S3 = 'Joe'); stop;
endprocess; endblock; endsystem;
```

Fig. 3. Person Entity with Employee and Student Roles in SDL-ARC

3 Coordination in the ARC Model

The entities within an application do not exist in isolation. Application features require collaborations among entities. A collaboration is a collection of roles played by participating entities that cooperate to realize an application feature. In the ARC model, an entity participates in collaborations through its role agents. The parallelism among role agents within a collaboration implies the need for coordination if the role agents are to cooperate effectively. Coordination in the ARC model is enacted by **coordinator** agents and is based on open and dynamic sets of role agents. Coordinator agents, like basic agents and role agents, are SDL-2000 process agents. Both coordinator agents and role agents may behave as **coordination** agents and react to events that occur within other observed agents. To enhance simplicity and locality, a role agent may observe only its parent agent, and a coordinator agent may observe only the current elements of its coordinated set(s) of role agents.

3.1 Coordinator Agent Behavior

A coordinator agent is the point of contact for an instance of a feature collaboration and also enacts coordination in reaction to events observed in coordinated role agents. The EFSM that specifies the behavior of a coordinator agent therefore handles signals from clients of a feature collaboration as well as signals from coordinated role agents collaborating within the feature. The signals sent to a coordinator from coordinated role agents may signal the following events.

- A coordinated role agent receives a specified signal from its input queue.
- A coordinated role agent transitions from a specified state.
- A coordinated role agent transitions into a specified state.

The coordination logic enacted by coordinator agents in response to events is state-based and may vary over time. The state of a coordinator agent consists of the explicit current state of its state machine and the implicit state stored in its local variables, including the open and dynamic sets of coordinated role agents.

The hybrid behavior and dual responsibilities of coordination agents distinguish the ARC model from actor-based coordination models in which coordination is enacted exclusively at the meta-level and direct communication among basic actors and meta-level coordination actors is impossible. The combination of role and coordinator agents in the ARC model facilitates modular coordination and enhances the modularity of features. A new application feature typically requires new role agents, new inputs for the respective parent agents to instantiate the new role agents, and a new coordinator agent to coordinate the role-based collaboration within the feature. The existing role agents of entities often remain oblivious to additional role agents. Modular coordination is facilitated if coordinator agents have the capability to send signals to coordinated role agents, since the logic to send the signals relevant in coordination scenarios may then be untangled from the behaviors of the coordinated role agents and specified cohesively within the behaviors of coordinator agents.

3.2 Coordination, Indivisibility, and Synchronization

The parallelism among role agents within a feature collaboration implies the need for coordination and synchronization of the role agents. In contrast to the explicit low-level synchronization primitives such as semaphores and condition variables that complicate parallel applications today, the ARC model enables synchronization to be expressed at a higher level of abstraction and implicitly implements low-level locking and mutual exclusion. The capability to atomically dispatch a set of signals to a set of agents is fundamental to many coordination scenarios. The SDL-ARC `dispatch` statement provides a novel synthesis of signal send and dispatch capabilities. A `dispatch` statement is executed only by a coordinator agent. The body of a `dispatch` statement must be a sequence of signal sends to a set of target role agents. The messages are atomically sent and dispatched using an adapted three-phase commit protocol.

1. In the first phase, all target agents are locked by the coordinator agent, ensuring that the target agents will dispatch only signals sent by the coordinator agent until the target agents are unlocked. Target agents are locked by the ARC model implementation sequentially and in the order determined by their respective identifiers (`Pids`). A lock on a target agent completes only when the target agent is in a state that defines an input for its associated target signal. Several deadlock and livelock scenarios are therefore avoided.
2. In the second phase, the sequence of signals is sent by the coordinator agent to the target agents, and the received signals are dispatched by the target agents. A run-time check ensures that only one signal is sent to each target agent within a given `dispatch` statement. Since the target agents are locked and sensitized to input signals only from the coordinator agent, the second phase must ultimately complete.
3. In the third phase, the target agents are unlocked in the order determined by their respective identifiers, execution of the `dispatch` statement is complete, and the coordinator agent continues execution after the `dispatch` statement with knowledge of the coordinated state of the target agents.

Indivisibility is a fundamental requirement of coordination. While a coordination event that ultimately impacts several coordinated agents is being processed, no intermediate states can be visible or accessible, and information required to process an event must remain valid throughout the event. When a coordinated agent triggers a coordination event by receiving a signal or executing a state transition, the coordinated agent may use the SDL-ARC `lock` statement to explicitly lock itself on behalf of a specified coordinator agent. The execution of `lock`(*coordinator*) immediately ensures that the coordinated agent will dispatch only signals received from the specified coordinator agent. When the coordinator agent attempts to lock the coordinated agent again during the first phase of a subsequent `dispatch` statement, the lock attempt will succeed without effect, and deadlock does not occur. Explicit locks are not distinguished during the third phase of a `dispatch` statement, and a target agent associated with an explicit lock is unlocked upon completion of the `dispatch` statement.

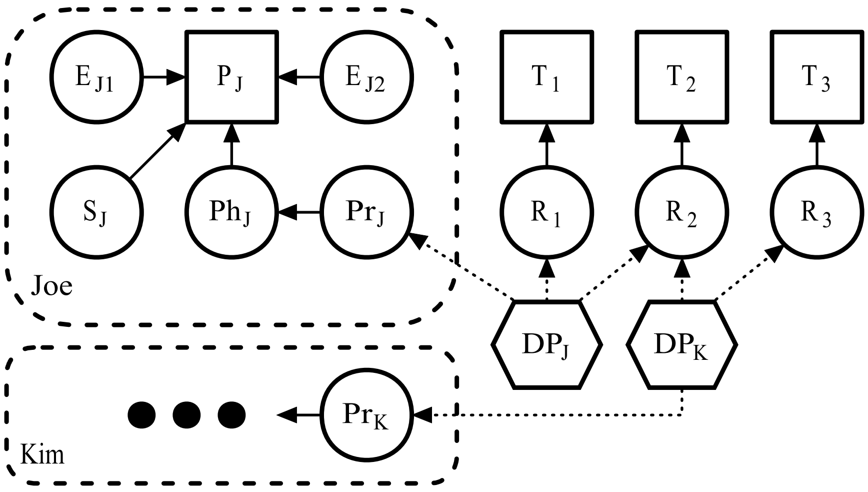


Fig. 4. The Dining Philosophers Problem in the ARC Model

3.3 Coordination Example

We illustrate coordination in the ARC model by solving a version of the dining philosophers problem. Each philosopher alternates between thinking and dining and must atomically acquire assigned tableware before dining and release the tableware after dining. While dining, each philosopher contributes to solving the NAS Embarrassingly Parallel (EP) benchmark [10]. The EP benchmark is typical of Monte-Carlo simulation applications in scientific computing and provides a concrete representation of a workload that is easily partitioned among ARC parallel agents. The ARC solution uses the roles of `Philosopher`, `Process`, and `Resource`, as illustrated in Fig. 4, and builds on the example entity illustrated in Fig. 2 and Fig. 3. In addition to the roles of `Employee` and `Student`, `Joe` also plays the `Philosopher` role in a collaboration of dining philosophers.

In Fig. 4, basic agents are denoted by squares, role agents are denoted by circles, and coordinator agents are denoted by hexagons. Solid arrows indicate parent relationships among basic and role agents. Dashed arrows indicate coordination enacted by coordinator agents. The ARC solution has the following structure.

- The intrinsic part of `Joe` is realized by a `Person` basic agent ($\mathtt{P_J}$).
- The extrinsic parts of `Joe` discussed earlier are realized by two `Employee` role agents ($\mathtt{E_{J1}}, \mathtt{E_{J2}}$) and a `Student` role agent ($\mathtt{S_J}$).
- The additional extrinsic part of `Joe` added for the dining philosophers problem is realized by a `Philosopher` role agent ($\mathtt{Ph_J}$) that plays the role of a process with resource requirements. A `Process` role agent ($\mathtt{Pr_J}$) realizes the process sub-role of the philosopher role played by `Joe`.
- The `Tableware` basic agents ($\mathtt{T}_i$) represent the resources (e.g. fork, spoon, chopstick) required by philosophers. The shared resource role played by each `Tableware` agent is realized by a `Resource` role agent ($\mathtt{R}_i$).

- A dedicated `DiningPhilosopher` coordinator agent coordinates the `Process` role agent for a philosopher and the `Resource` role agents required by the philosopher, e.g. (DP_J) is dedicated to `Joe`. Note that each `Resource` role agent is coordinated by multiple `DiningPhilosopher` coordinator agents, reflecting the inherent resource contention in the problem.

Before considering the required coordination, we summarize the behaviors of the `Tableware`, `Person`, and `Philosopher` agents specified in Fig. 5.

```
system DiningPhilosophers; block B;
syntype Agent = Pid;
syntype uint8_t  = Integer constants (0 :   255);
syntype uint16_t = Integer constants (0 : 65535);
value type SetAgent inherits String<Agent>;
value type VectorEP inherits Vector<Real,12>;
signal EPresult(VectorEP),
  acquire(SetAgent), acquire$, release, release$, new(Agent),
  asPhilosopher(uint8_t, uint8_t), asProcess(Agent), asResource;
process Process referenced; process Resource referenced;
process DiningPhilosopher referenced; process Main referenced;

process Tableware (0,255);
signalset asResource; dcl resource Agent := Null;
start; nextstate start$; state start$;
  input asResource; task {
    if (resource = Null) { create Resource; resource := offspring; };};
    output new(resource) to sender; nextstate -;
endprocess;

process Person (0,255) (name, phone Charstring);
signalset asPhilosopher; dcl number, units uint8_t;
start; nextstate main; state main;
  /* previous behavior assumed but omitted here to conserve space */
  input asPhilosopher(number, units); create Philosopher(number, units);
    output new(offspring) to sender; nextstate -;
endprocess;

process Philosopher (0,255) (number, units uint8_t);
signalset asProcess, acquire;
dcl tableware SetAgent, coordinator Agent, V VectorEP; timer T := 10.0;
procedure EP(in uint8_t, in/out uint8_t, in/out VectorEP) external;
start; nextstate start$; state start$;
  input asProcess(coordinator); create Process(coordinator);
    output new(offspring) to sender; set(T); nextstate thinking;
state thinking; input T;
  decision units = 0;
    (true) : output EPresult(V) to coordinator; stop;
     else  : nextstate hungry;
  enddecision;
state hungry; input acquire(tableware); nextstate dining;
state dining; provided true;
  call EP(number, units, V); set(T); nextstate thinking;
endprocess;
endblock; endsystem;
```

Fig. 5. The Dining Philosophers Problem in SDL-ARC (part 1)

Each `Tableware` basic agent creates a single shared `Resource` role agent and outputs the agent in response to the `asResource` input. The `Person` basic agent adds the `asPhilosopher` input in addition to the behavior discussed above. The behavior of the `Philosopher` role agent is expressed as the following EFSM.

- In the start$ state, the philosopher assumes the Process role in response to the asProcess input, sets the timer (T) to expire after an interval of thinking, and transitions to the thinking state.
- In the thinking state, after the timer (T) expires, the philosopher decides whether its contribution to the EP benchmark has been completed. If all units of work have been completed, the philospher outputs its contribution and stops; otherwise, the philosopher transitions to the hungry state.
- In the hungry state, the philosopher awaits the input of an acquire signal with the appropriate set of tableware. The philosopher transitions to the dining state after acquiring its tableware.
- In the dining state, the philosopher completes one unit of benchmark work via the EP procedure, sets the timer (T), and returns to the thinking state.

The additional coordination agents specified in Fig. 6 include the Process, Resource, and DiningPhilosopher agents. The coordination enacted by the DiningPhilosopher coordinator agent is triggered by a sequence of events that occur in the Philosopher and Process role agents. Coordination in the ARC model is treated as an orthogonal concern, and established aspect-oriented programming technology is leveraged to help enact coordination. SDL-ARC aspects are influenced by AspectJ [11] and are composed of join points, pointcuts and advice. However, join points in SDL-ARC are customized to SDL and include agent instantiation, agent termination, state inputs, and state transitions in the EFSM behaviors of agents. We believe that aspects based on EFSM behavior are more abstract and less fragile than aspects based on lower-level control flow. Two aspects are specified in Fig. 6.

The PhilosopherProcess aspect observes events in the Philosopher role agent. The perAgent declaration ensures that an aspect instance is dedicated to each Philosopher agent. The aspect has three pieces of advice.

- After a Philosopher agent creates a Process agent, the identity of the offspring is saved in the process$ variable.
- After a Philosopher agent transitions into the hungry state, the acquire$ signal is output to the saved Process agent.
- After a Philosopher agent transitions from the dining state, the release$ signal is output to the saved Process agent.

The DiningPhilosopherProcess aspect observes events in the Process role agent. The singleton aspect has two pieces of advice that use the coordinator parameter of a Process agent to output signals to a DiningPhilosopher agent.

- After a Process agent transitions into the acquire state, the acquire$ signal is output using the coordinator parameter.
- After a Process agent transitions from the release state, the release$ signal is output using the coordinator parameter.

The acquire$ input sent from the PhilosopherProcess aspect causes the Process role agent to transition from its start$ state into its acquire state.

```
aspect PhilosopherProcess perAgent(Philosopher());
  pointcut Philosopher() : self(Philosopher); dcl process$ Agent;
  after() : Philosopher() and create(Process)
  { task process$ := offspring; }
  after() : Philosopher() and state+(hungry)
  { output acquire$ to process$; }
  after() : Philosopher() and state-(dining)
  { output release$ to process$; }
endaspect;

aspect DiningPhilosopherProcess;
  pointcut Process() : self(Process);
  after() : Process() and state+(acquire)
  { output acquire$ to Process.coordinator; }
  after() : Process() and state-(release)
  { output release$ to Process.coordinator; }
endaspect;

process Process (0,255) (coordinator Agent);
signalset acquire, acquire$, release$; dcl tableware SetAgent;
start; nextstate start$;
state start$;  input acquire$; nextstate acquire;
state acquire; input acquire(tableware);
               output acquire(tableware) to parent; nextstate release;
state release; input release$; nextstate start$;
endprocess;

process Resource (0,255); signalset acquire, release;
start; nextstate released;
state released; input acquire; nextstate acquired;
state acquired; input release; nextstate released;
endprocess;

process
DiningPhilosopher (0,255) (philosopher Agent, tableware SetAgent);
signalset acquire$, release$, new, EPresult;
dcl process$, resource SetAgent := emptystring, V VectorEP;

procedure newProcess(philosopher, coordinator Agent) -> Agent;
/* create Process agent with philosopher as parent */ endprocedure;
procedure newResource(tableware Agent) -> Agent;
/* create Resource agent with tableware as parent */ endprocedure;

start; task { process$ := mkstring(newProcess(philosopher, self));
  loop (dcl t uint8_t := 1, t <= length(tableware), t+1)
    resource := resource // mkstring(newResource(tableware(t)));
  };
  nextstate main;
state main;
  input acquire$;
    dispatch {
      output acquire(tableware) to process$(1);
      loop (dcl r uint8_t := 1, r <= length(resource), r+1)
        output acquire(emptystring) to resource(r);
    };
    nextstate -;
  input release$;
    dispatch {
      loop (dcl r uint8_t := 1, r <= length(resource), r+1)
        output release to resource(r);
    };
    nextstate -;
  input EPresult(V); output EPresult(V) to parent; stop;
endprocess;
```

Fig. 6. The Dining Philosophers Problem in SDL-ARC (part 2)

This state transition triggers an input at the DiningPhilosopher coordinator agent (explained below) that will ultimately result in the dispatch of an acquire input to the Process role agent with the appropriate set of tableware. The Process role agent forwards the acquire input with the set of tableware to its parent Philosopher role agent and transitions into its release state. The release$ input sent from the PhilosopherProcess aspect causes the Process role agent to transition from its release state back into its start$ state, triggering another input at the DiningPhilosopher coordinator agent.

The DiningPhilosopher coordinator agent defines two sets of coordinated role agents. The process$ set is a singleton set whose member is the Process role agent created by the Philosopher role agent specified in the philosopher parameter. The resource set is the set of Resource role agents associated with the Tableware basic agents required by the Philosopher role agent. The DiningPhilosopher coordinator agent reacts to the acquire$ input sent from the DiningPhilosopherProcess aspect by atomically dispatching acquire messages to the Process role agent and its required Resource role agents. The acquire message sent to the Process role agent contains the required tableware to be forwarded to the Philosopher role agent. Each of the locks on the Resource role agents attempted sequentially during the first phase of the dispatch statement completes only when the Resource role agent is in its released state, ensuring the coherence and correctness of the atomic acquisition of resources. The DiningPhilosopher coordinator agent reacts to the release$ input sent from the DiningPhilosopherProcess aspect by atomically dispatching release messages to the previously acquired Resource role agents.

The EFSM behavior of the Main agent is simple but is not specified here because of space restrictions. To maximize parallelism in the EP benchmark, the Main agent creates twice the number of Philosopher and Tableware agents as the number of threads (cores) available, since at most half of the Philosopher agents can be dining concurrently. A dedicated DiningPhilosopher agent is created to coordinate each Philosopher agent and its required Tableware agents. The Main agent also accumulates the EP benchmark contributions from the Philosopher agents, verifies the accuracy of the total, and prints a report.

In the SDL-ARC version of the dining philosophers problem, a Philosopher role agent has explicit knowledge of acquired Tableware agents only in the dining state, and the set of acquired Tableware agents could change at each transition into the dining state. An enhanced DiningPhilosopher coordinator agent could dynamically adjust the open sets of Tableware agents and associated Resource role agents required by a Philosopher agent and its Process role agent. The required coordination is modular, enabling each role agent EFSM behavior to be concisely expressed and oblivious to the myriad coordination details inherent in the atomic acquisition of a dynamic set of shared resources. Implicit locking and the avoidance of some deadlock and livelock scenarios enhance the quality of pervasive applications and the productivity of software engineers.

4 Realization of the ARC Model

An implementation of SDL-2000 that is fully conforming, fully functional, and available to the public is not yet readily available. SDL-ARC models are currently translated by hand to equivalent SDL-96 models. In addition to the documented SDL-2000 to SDL-96 conversion [3], the SDL-ARC extensions (`dispatch` and `lock` statements, aspect-oriented advice) must also be implemented during the translation. An SDL-ARC compiler that automates the translation to SDL-96 is currently in development.

The implementation of the PSM for an SDL-ARC PIM uses a thread pool pattern in which the number of application worker threads is closely matched to the available processors. The conceptual threads dedicated to parallel agents are mapped onto the worker threads by the PSM implementation. An agent is scheduled in an available application thread when a signal is input to the agent, and the agent is run to completion on the input signal in the assigned thread.

The version of the dining philosophers problem presented above was admittedly selected to illustrate the potential of the ARC model. The exploitation of multicore parallelism in arbitrary MDE applications will undoubtedly be more challenging. Nevertheless, the table below illustrates the performance achieved when comparing the SDL-ARC PIM of the dining philosophers EP benchmark to the NAS serial version of the EP benchmark written in C. The target platform was a Sun UltraSPARC T2 1.2 MHz processor with 8 multithreaded cores. The efficiency measures how well the SDL-ARC PSM implementation utilizes the multicore parallelism, including the overhead due to communication and synchronization. Efficiency is defined by the formula $E = T_1/(T_t * t)$, where T_1 is the execution time of the NAS serial version, and T_t is the execution time of the SDL-ARC parallel version using t threads and $(t * 2)$ philosophers. The efficiency degrades beyond 16 philosophers (8 threads), presumably because of intra-core resource contention on the UltraSPARC T2 introduced by multiple threads sharing the single floating point unit (FPU) on each of the 8 cores.

Execution Time (seconds) and Efficiency (%), phil=philosophers

input size	NAS serial	SDL phil=2	SDL phil=4	SDL phil=8	SDL phil=16	SDL phil=32	SDL phil=64
2^{28}	293.640s	296.050s	150.900s	74.980s	38.870s	24.640s	13.450s
2^{28}	N/A	99.19%	97.30%	97.91%	94.43%	74.48%	68.22%
2^{30}	1176.688s	1183.120s	590.585s	303.678s	155.530s	80.453s	44.452s
2^{30}	N/A	99.46%	99.62%	96.87%	94.57%	91.41%	82.72%

5 Related Work

The initial publication of the ARC model [12] discussed related work in the area of coordination in some depth. The ARC model has since been generalized and retargeted to SDL-2000. The coordination enacted in the ARC model remains event-based, role-based, state-based, open, and dynamic. In the updated ARC model, the logic specified in coordinator agents combines both the computation and the coordination required by a feature. However, the collaboration-based coordination in the ARC model is modular in the presence of cross-cutting features and untangled from the behaviors of role agents and basic agents.

The goals of the ARC model and the Epsilon model [13] are similar, and the survey of work related to the Epsilon model is of interest here. The Epsilon model enables object-oriented delegation via the export of features both to and from role objects, while in the ARC model, features are only exported from parent agents to role agents. This ARC model design decision aligns with established practice in role modeling and minimizes semantic changes to SDL-2000.

The SDL-2000 standard introduced several constructs that facilitate the specification of structural hierarchy, including generalized nested agents, composite states, and state aggregations. A composite state is a nested state machine and may also be a state aggregation. A state aggregation is composed of partitions (each with a state machine) that handle disjoint sets of input signals, and a state aggregation is in one state in each of its partitions. An alternate realization of a role-based entity might use an agent whose state machine is a state aggregation, where each partition of the state aggregation corresponds to a role. However, the state aggregation semantics specifies interleaved scheduling that enables exactly one partition to be executing at any point. The alternate realization requires a more complicated implementation that enables parallelism among partitions only when the intuitive behavior required by interleaved scheduling can be maintained, potentially adding overhead and limiting intra-entity parallelism.

6 Conclusion and Status

The computational landscape is being shaped by multicore processors and pervasive computing. The ARC model leverages existing software technology to address the question of how to program software applications that are open, distributed, and parallel. The foundation for the ARC model is SDL-2000, which embodies decades of experience in building distributed systems, maps directly to the Actor model, aligns with the principles of model-driven software development, and enables relatively abstract platform-independent parallelism. However, SDL-2000 does not offer a detailed model for structuring computation among process agents that exposes natural opportunities for parallelism. The ARC model uses a synthesis of role-oriented modeling and the split object model to structure computation in terms of conceptual entities. An entity is composed of a basic intrinsic agent and multiple extrinsic role agents, all with dedicated conceptual threads of control. Entities in the ARC model interact through their role agents in the context of feature-oriented collaborations with dedicated coordinator agents. The basic agents, role agents, and coordinator agents in the ARC model are SDL-2000 process agents. Role agents and coordinator agents are coordination agents that use aspects to observe events in other agents. The coordination logic enacted by coordinator agents uses the capability to atomically send and dispatch messages to coordinated role agents in response to events.

The experience with hand-translated SDL-ARC examples has been positive. We plan to continue working with examples from the literature to identify opportunities to apply the modular coordination and abstract parallelism in the ARC model. Additional extensions to SDL-ARC that more concisely express ARC model usage patterns will be investigated. The completion of the SDL-ARC compiler will enable us to assess the utilization of multicore processors by significant SDL-based MDE applications and evaluate ARC model design decisions made to this point.

References

1. Schmidt, D.C.: Model-driven engineering. IEEE Computer 39(2), 25–31 (2006)
2. Mellor, S.J., Kendall, S., Uhl, A., Dirk Weise, D.: MDA Distilled. Addison-Wesley, Reading (2004)
3. International Telecommunications Union: Recommendation Z.100 (11/07), Specification and Description Language (SDL) `http://www.itu.int/rec/T-REC-Z.100/en`
4. Object Management Group (OMG): UML Superstructure Specification, Vers. 2.1.2, `http://www.omg.org/docs/formal/07-11-02.pdf`
5. International Telecommunications Union: Recommendation Z.109 (06/07), SDL combined with UML, `http://www.itu.int/rec/T-REC-Z.109/en`
6. Ellsberger, J., Hogrefe, D., Sarma, A.: SDL — Formal Object-Oriented Language for Communicating Systems. Prentice-Hall, Englewood Cliffs (1997)
7. Agha, G.: Actors: A Model of Concurrent Computation in Distributed Systems. MIT Press, Cambridge (1986)
8. Kristensen, B.B.: Object-oriented modeling with roles. In: Murphy, J., Stone, B. (eds.) Proceedings of the 2nd International Conference on Object-Oriented Information Systems, pp. 57–71 (1996)
9. Bardou, D., Dony, C.: Split objects: A disciplined use of delegation within objects. ACM SIGPLAN Notices 31(10), 122–137 (1996)
10. Bailey, D.H., et al.: The NAS Parallel Benchmarks. The International Journal of Supercomputer Applications 5(3), 63–73 (1991)
11. The AspectJ Project, `http://www.eclipse.org/aspectj/`
12. Ren, S., Yu, Y., Chen, N., Marth, K., Poirot, P.-E., Shen, L.: Actors, roles and coordinators - a coordination model for open distributed and embedded systems. In: Ciancarini, P., Wiklicky, H. (eds.) COORDINATION 2006. LNCS, vol. 4038, pp. 247–265. Springer, Heidelberg (2006)
13. Tamai, T., Ubayashi, N., Ichiyama, R.: An Adaptive Object Model with Dynamic Role Binding. In: ICSE 2005, pp. 166–175. ACM, New York (2005)

Automatic Generation of Compatible Interfaces from Partitioned UML Activities

Frank Alexander Kraemer

Norwegian University of Science and Technology (NTNU),
Department of Telematics, N-7491 Trondheim, Norway
kraemer@item.ntnu.no

Abstract. Behavioral interfaces describing the sequences of buffered signal transmissions between components are useful to ensure that components can interact consistently. Previous work has shown that compatible interfaces at each side of a channel are in general asymmetric: In cases where both communicating sides can take initiative and send, they have to be prepared for conflicts that need resolution. Such conflicts are often difficult to handle properly at the interaction level, where the intent of the actual application may be obscured by technical details. For this reason, we developed an automated method that starts with a holistic and collaborative view on distributed behavior provided by UML activities, in which components are represented by separate partitions. Once the activity is well-formed, pairs of interfaces for each channel are derived by cutting the activity along its partitions and exploring the visible behavior between them. The resulting interfaces are compatible by construction and allow other entities to interact consistently across them. The proposed method is implemented as part of our tool support.

1 Introduction

One principle in service-oriented approaches is that systems do not have to be specified and implemented at once and by one stakeholder, but that they can be provided incrementally, with new components introduced dynamically at run-time. Obviously, in order that new components can make use of the functionalities already provided or replace existing components, they need to interact consistently with the other components. A key concept is therefore the definition of interfaces between components. In a number of approaches, such interfaces are called *services* and specify only the operations provided or required by a component, corresponding to signals sent and received in an asynchronous communication setting. However, to ensure consistent interactions, the sequences, in which messages are to be exchanged, are also important. But while this has been discussed in many approaches in literature (see, for example [6,7,8,11,22]), it seems that such interfaces are not yet adopted by programmers for everyday development. As one reason for that we see the level of expertise and effort needed to design and understand such interfaces in a correct way with all their intricacies. When highly concurrent systems are designed using a buffered communication scheme (as offered for example by enterprise service busses), the delay of signals has to be taken into account. This leads to interfaces that are more complicated than the naïve solution that a developer often would initially develop.

Some approaches find a solution to this problem in restricting interactions to certain patterns that are easier to handle (for instance simple request/response patterns), and avoid the usage of any elaborate protocols in the first place. We think such solutions are unfortunate, since they prevent a system from executing according to its full potential [25]. Furthermore, basic protocols can also be intricate. This paper shows how even a rather simple subscription-based pattern requires asymmetric interfaces, due to a conflicting initiative between the regular notification and the command to unsubscribe from the client. Failing to treat these cases with the necessary precision will in the best case lead to an unsharp specification style where it is not clear if a certain behavior is intended by the designer or whether it is a flaw. In the worst case it leads to interfaces that will not work correctly in any environment.

To improve this situation, we propose a technique in which compatible interfaces are synthesized *automatically* from a higher-level description expressed in form of a *collaboration* with behavior defined using UML activity diagrams. This is a description of behavior among several participants, including their interactions and those parts of their local behavior that are relevant for the task of the collaboration. Since activities express complete behavior, they can be model checked, so that the soundness of the collaboration can be ensured. With the tools described in [15], we show how this can be ensured in a rather automated process, where flaws are explained to users by animations. Once a collaboration is sound, our algorithm uses the state space obtained from the analysis tools to construct the interfaces for each communication partner and encapsulates them as service contracts. As a result, developers do not have to develop pairs of compatible interface descriptions manually, but may focus on the development of a sound collaboration suitable for their application. This task is usually easier to understand, since it is closer to the actual application and intricacies such as mixed initiatives are handled by the algorithm constructing the interfaces.

The generation mechanism for interfaces described here is an important element in a more comprehensive framework for the dynamic provision and discovery of services. Within this framework, services are provided as collaborations, as described later. Functions like registration, discovery and optional negotiation of services are contributed as dedicated, collaborative building blocks. This facilitates the construction of dynamic services in a specification-driven way. In the following, we focus on the derivation of service contracts from activities and present the example of a simple SMS notification protocol, in which a receiver can subscribe to an SMS gateway to receive messages that are sent by mobile user to a certain number. Based on the example, we state the criteria for sound interactions and develop the interfaces for the example. In Sect. 3 we introduce our collaboration-oriented engineering method SPACE by giving a collaboration for the SMS notification and their detailed semantics. The algorithm is introduced in Sect. 4 and discussed in Sect. 5.

2 Behaviored Interfaces and Service Contracts

As an example, we assume to be a telecom operator who wants to make an SMS Notification service accessible to third-party service providers. With this service, third parties can receive all SMS messages that are sent by mobile phones to a certain number. For that reason, we build an SMS gateway C_g that can be accessed by clients like the SMS receiver C_r, as shown in our first figure.

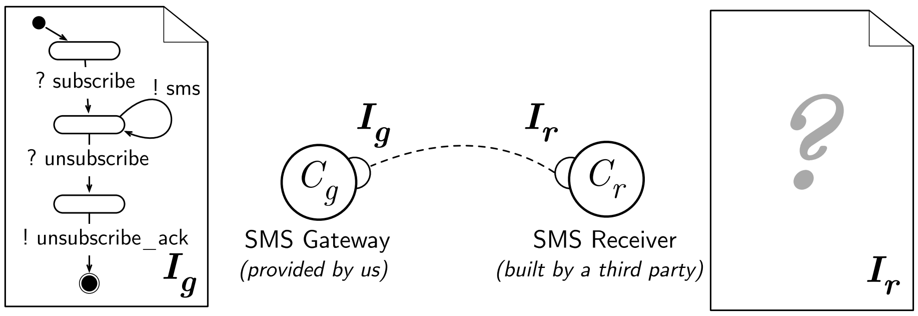


Fig. 1. SMS Notification service provided to clients

The signal exchanges for the service are described by interface I_g. It is started by the reception (*?*) of a *subscription* message sent by the receiver. From then on, the SMS gateway forwards all incoming SMS to the receiver, expressed by the sending (*!*) of message *sms*, until the receiver unsubscribes, which is acknowledged by message *unsubscribe_ ack*.

2.1 Interfaces Come in Pairs of Two

When a third party wants to build an SMS receiver interacting consistently with the SMS gateway, a naïve look at interface I_g may lead to an erroneous solution: Programmers may forget that even if they have sent *unsubscribe* to the gateway, they still must be prepared to receive an *sms* from the gateway. This is because I_g is the interface *local to the gateway*, and we assume that the communication between the receiver and the gateway is buffered. Therefore, two messages can cross their ways and lead to conflicting behaviors. Such a situation is also called *mixed* [11] or *conflicting* [5] initiative, or *non-local choice* [4]. For the SMS notification, the receiver therefore has to be prepared to accept an SMS message even after it sent *unsubscribe*.[1] For this reason, an interface local to the receiver, I_r, should be considered instead when building the receiver. Since any receiver interested in the gateway can make use of this interface, we agree with the authors of [27] that interfaces should be provided as pairs.

[1] Internally the receiver can still deliberately drop the SMS message, but such a decision should be done explicitly by an engineer with knowledge about the application.

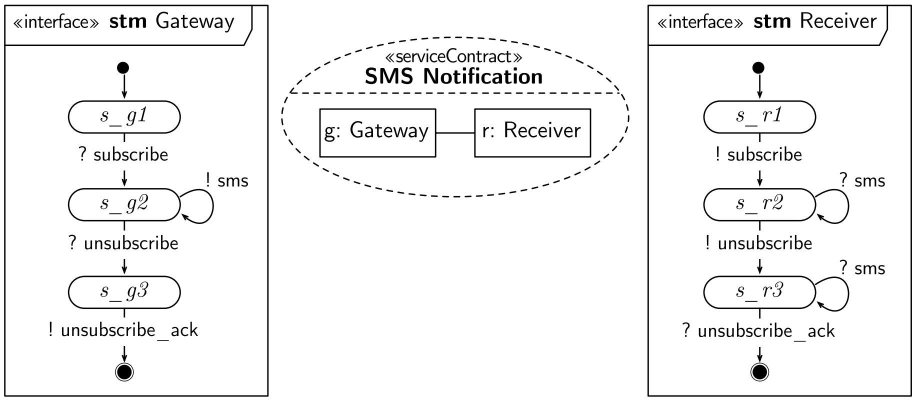


Fig. 2. Interfaces for receiver and gateway for the SMS Notification

For the given example, Fig. 2 shows the pair of interfaces I_g and I_r, with I_r being local to the receiver. We note that I_r is not a simple inversion of I_g, but structurally different due to the reception of *sms* in state *s_r3*. In the center of the figure, a stereotyped collaboration is shown. It is a *service contract* in line with the emerging SoaML standard [24] and refers with its two collaboration roles g and r to UML state machines describing both interfaces.

The question remains how to build compatible interfaces, such as I_g and I_r, in the first place. While the example above may seem trivial to some, such intricacies of buffered communication are in our experience hard to impart to programmers. Furthermore, once several conflicts can happen, or interfaces describe more than one ongoing concurrent functionality, the design of compatible interfaces I_g and I_r can get quite difficult also for experts. Before we come to the generation of such interfaces, let us first consider the criteria for consistent interactions.

2.2 Interface Compatibility

An interface I_i is a state machine attached to the end of a communication channel at the component it represents to describe the allowed sequences of signal sendings and receptions. In the following, we assume error-free and order-preserving channels. Starting in its initial state, an interface updates its current state with every sending and receiving action of its component over the channel. A pair of interfaces is compatible if and only if two communication partners that obey the behavior described by their respective interfaces interact soundly. This means that:

(i) There are no unspecified receptions, i.e., whenever a signal arrives at an interface from a communication partner, the interface is in a state that enables a transition allowing the reception of that signal.
(ii) There are no deadlocks, i.e., when an interface is in a state waiting only for a signal reception from the communication partner, it will eventually receive such a signal.

In [11], properties (i) and (ii) are also called *containment* and *obligation*, respectively. The reception of the additional *sms* of the receiver in Fig. 2 in in state *s_r3* can be generalized as a rule for interfaces in order to be able to interact consistently with them: An interface declaring outgoing sending and receiving transitions in a state s_1 must declare also all receiving transitions in all states s_i that are targeted via sending transitions from s_1. This is called *input consistency* [5].

With the algorithms presented in [10,11,25], the compatibility of two interfaces with respect to criteria (i) and (ii) can be checked. Before that, however, the interfaces have to be created in the first place.

3 Focus on Collaborations

In Sect. 4, we present an algorithm that generates service contracts from the more holistic specifications provided by *collaborations*, that means behavioral descriptions comprising several participants. For that, we first briefly describe our engineering method to set the larger context of our construction algorithm, and then develop the SMS notification from a collaborative perspective.

3.1 The SPACE Engineering Method

The idea behind this method [14,16] is to use collaborations as the major specification units, which can be stored in a variety of libraries for different domains, as shown on the left side of Fig. 3. Since these building blocks can cover behavior among several components, they facilitate the reuse of solutions that require several participants to be coordinated, which means that entire sub-services, including all their participants, can be reused. During service and system design, blocks can be taken from such libraries and composed together. If no block exists yet, new ones may be created from scratch and stored in a library for later reuse.

For the behavioral description of the blocks we use UML activities. These activities can have input and output parameters, which can later be used for composing activities together and pass data or synchronize events. With an

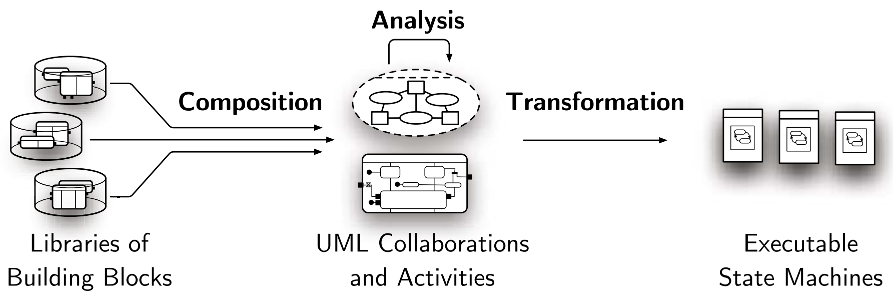

Analysis
Composition
Transformation
Libraries of Building Blocks
UML Collaborations and Activities
Executable State Machines

Fig. 3. The SPACE method based on reusable, collaborative building blocks

additional UML state machine, a so-called ESM, the externally visible behavior of these parameters is specified. This makes it possible to reuse a collaboration without looking at the details of its internal behavior; the ESM is sufficient to compose it correctly. The composition of the blocks is analyzed by means of model checking. Due to the ESMs, a building block can always be analyzed in isolation; any constituent blocks are abstracted by their ESMs, which reduces the state space required during the analysis. Once a system is complete and sound, it may be implemented. For that, we use a process consisting of two steps: In the first step, we automatically transform the activities into executable UML state machines (similar to SDL processes, see [19]), using the algorithm presented in [18]. In a second step, code is generated from the state machines [14].

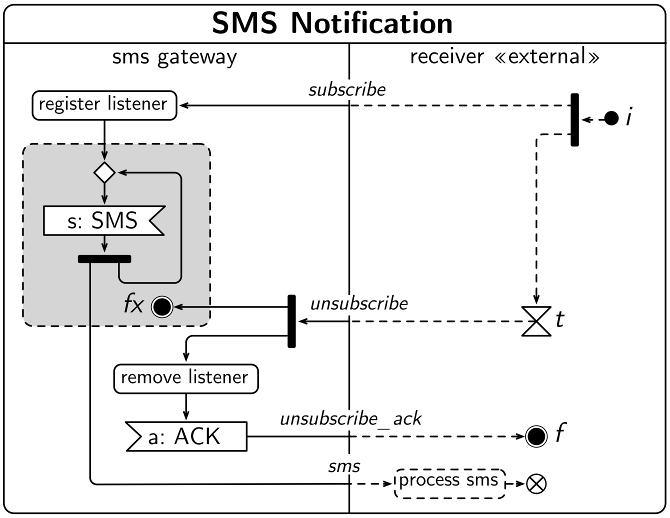


Fig. 4. Activity describing both sides of the SMS notification

3.2 Collaboration for the SMS Notification

Fig. 4 shows the behavior of the collaboration for the SMS notification. It consists of two activity partitions; one for each participant of the collaboration: the SMS gateway and the receiver. Since we assume that we are the developers of the SMS gateway, the activity shows all necessary actions needed for the SMS gateway, which we will use to implement the server via our transformation and code generators. We assume that the receiver, on the other hand, will eventually be developed by a third party. Therefore, the partition for it is marked as «external», and its behavior is just sketched. In particular, the timer *t* represents any spontaneous initiative from the receiver side to unsubscribe. The detailed behavior of this activity will be presented in the next section, after a brief introduction to the formal semantics for activities.

3.3 Formal Semantics in Temporal Logic

To ensure the correctness of the method and define the semantics of all diagrams, we reason in temporal logic, in particular the Temporal Logic of Actions (TLA, [21]). This is a linear-time temporal logic and describes behaviors as infinite sequences of states, where each state is an assignment of values to variables. Steps between states are described by actions, which effectively describe preconditions and effects on variables. Using cTLA [12], a compositional variant of TLA that introduces processes and their composition by joint actions [3], we have formalized the behavior of collaborations using the special dialect cTLA/c [17]. It assigns TLA actions to one of each collaboration participant, introduces queue variables between participants for communication and defined rules for the usage of these variables by the actions to ensure realizability. Therefore, it is ideal for the description of the behavior as expressed by UML activities. Since activities have semantics close to Petri nets, they describe state transition systems, which describe the movement of data or control tokens along the activity graph. The token markings (the places on which the tokens rest) are expressed by variables, and the token movements are expressed by TLA actions that remove and add tokens at the corresponding variables.

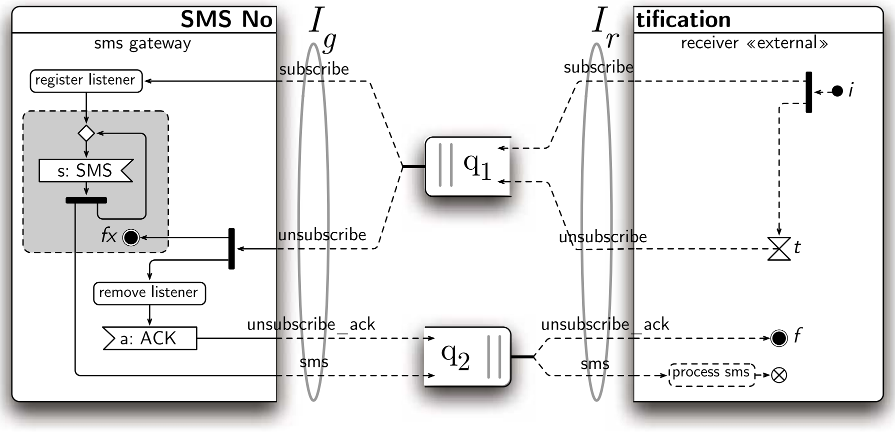


Fig. 5. Transmission queues between the activity partitions

In our example for the SMS notification, we use the variables i for the initial node, t for the timer, s for the accept signal node for SMS and a for the accept signal node waiting for an ACK.[2] Moreover, to respect the delays of the buffered communication, we add queue variables where flows cross the partitions. Queue q_1 is used by all tokens flowing from the receiver to the gateway, and queue q_2 is used by tokens flowing in the opposite direction. We describe queues as

[2] Activity final node *fx* doesn't need a variable since it never holds any tokens; when a token arrives to *fx*, the structured node around it is terminated and the token from *s:SMS* is removed.

sequences of signals $q_i = \langle sig_0, ..sig_n \rangle$, with $sig_i \in \{\mathsf{subscribe}, \mathsf{unsubscribe}\}$ for q_1 and $sig_i \in \{\mathsf{sms}, \mathsf{unsubscribe\_ack}\}$ for q_2. The queues are located between the activity partitions, illustrated in Fig. 5.

In [17], we described how the detailed TLA actions can be derived from an activity. But instead of listing the formulas for the SMS notification, we illustrate each action graphically by showing the affected sub-graph of the activity in Fig. 6. A token added is illustrated by ⊕, a removed token by ⊖. Tokens added to queues are appended at the tail, and tokens must be at the head position in order to be removed. Actions $r_1..r_4$ are executed by the receiver, and actions $g_1..g_4$ by the gateway.

3.4 Analysis of the Collaboration

With the automated analysis tool presented in [15,29], we make sure that the collaboration for the SMS notification fulfills a number of general properties that should hold for any application:

– A collaboration should terminate consistently, that means in such a way that all queues are empty once the activity reaches an activity final node.
– No deadlocks must occur, that means there must not be a situation in which no action can be executed but the activity final node has not yet been reached.
– If a collaboration is composed from other building blocks, these blocks must be used according to their external description.
– If a collaboration is encapsulated by an ESM, its internal behavior given by the activity must conform to the external contract described by its ESM (not shown here).
– The queues between activity partitions must be bounded.

Once problems are detected, error situations are presented as animations within the activity [15], and in some cases, improvements are suggested [29]. For the example, the analysis detects that the number of tokens in queue q_2 grows beyond a certain value, indicating unbounded behavior. For this case, [29] describes several means to ensure boundedness. For the example, we use an estimate on the timing behavior of the system: We know, that the arrival of SMS messages happens slowly compared to the transmission time of signals and the subsequent processing by the receiver. Technically, we can assert this in the specification by a stereotype on the activity edge represented by queue q_2. The respective action is then enabled only when q_2 does not already contain the maximum number of sms tokens. Another possibility is the introduction of an explicit buffer managed by the gateway and an additional confirmation of each SMS message by the receiver.

3.5 Collaborations and Service Contracts

If we just want to build the SMS gateway and one specific receiver, a collaboration that describes the behavior of each participant by a partition of a UML

Action r_1: The initial action is started by a token emitted from the initial node. After the duplication by the fork node, the timer is started and one token is placed in queue q_1.

Action g_1: The gateway receives the subscription, registers its listener and starts listening for internal signals.

Action g_2: Once the gateway's receives an SMS from the server logic, a token is emitted by accept signal action s. With the subsequent fork, the token is sent towards the receiver (via queue q_2) and one token is placed back into s, to continue listening.

Action r_2: The client receives the SMS from q_2. Here the processing of the message is only sketched by operation *process sms*. The subsequent flow final node just consumes the token.

Action r_3: Once the receiver does not want to receive SMS anymore, it moves a token *unsubscribe* into q_1. We trigger this action by timer t, which represents any spontaneous initiative of a real client.

Action g_3: The gateway receives the signal for unsubscribing. The token flowing into activity final node *fx* removes the token from the accept signal action s, due to the structured node around it. Furthermore, the listener is removed, and we start waiting for the confirmation *ACK* of the server logic.

Action g_4: Once the acknowledgement arrives via the internal *ACK* from the gateway logic, a corresponding token is placed into queue q_2.

Action r_4: The receiver consumes the acknowledgement. (A real receiver would then terminate or invoke further behavior.)

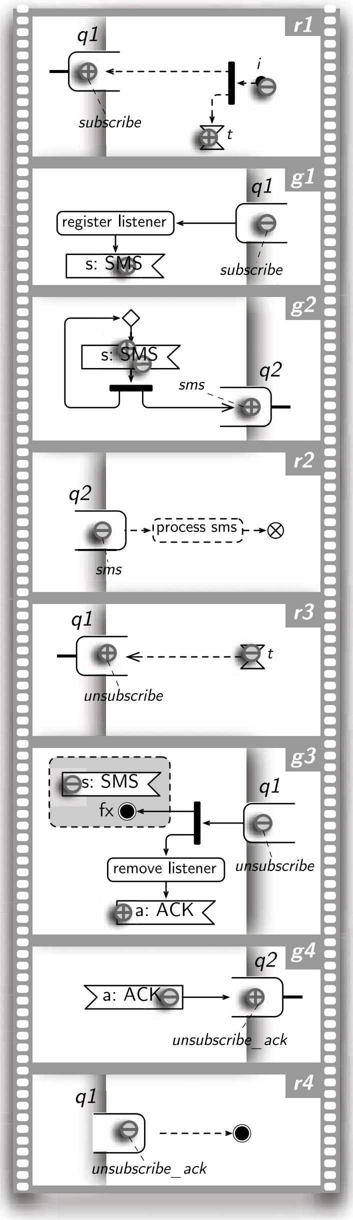


Fig. 6. Detailed actions defining the behavior of the SMS Notification

activity as in Fig. 4 is sufficient, since we can generate executable components directly from this representation [18]. For a dynamic scenario, in which new components can join the system at run-time and then use and provide services such as the SMS notification, we utilize service contracts as introduced in Sect. 2. In our collaborations, these service contracts describe the visible behavior in terms of signal transmissions *between* the activity partitions. For the example, the service contract consists of the two interfaces I_g and I_r. These are located at the partition borders, illustrated in Fig. 5, and describe the sequence of tokens (resp. signals once implemented) entering or leaving the partition.

4 Synthesizing Service Contracts from Activities

Figure 7 illustrates the steps needed to construct a pair of interfaces from an activity. We start with an activity describing both participants by dedicated partitions, such as the SMS Notification presented before. We assume that the activity is sound with respect to the criteria given in Sect. 2.2. In a first step, we obtain the state space of the behavior implied by the activity via model checking. From the resulting graph we extract one graph for each interface by labeling the steps according to the observable signal transmissions at the corresponding partition border. In two further steps, silent steps are removed, and the graph is minimized. The resulting graphs are used to construct UML state machines as the interface description, which are paired in a collaboration to form the complete service contract.

4.1 Step 1: Generating the State Space of an Activity

With the definition of all actions $r_1..r_4$ and $g_1..g_4$ from Sect. 3.3 and their effects on the variables i, t, a, s and the signal queues, we calculate the state space of the behavior. For that, we start with the initial state of the activity, in which only the initial node contains one token, and determine all reachable states by executing the actions for each state they are enabled.

For the SMS Notification, the complete state space is shown in Fig. 8. Each state $m_0..m_{11}$ corresponds to a specific token marking in the activity, and all edges refer to the actions of Sect. 3.3 that are executed. The graph is built using the assertion for bounded queues in Sect. 3.4 for at most one sms token in q_2. To the right of the graph, the detailed values of the variables are given for the markings m_0 to m_3 in TLA$^+$ syntax [21]. For proceeding further, these are not important, since the interfaces are constructed only from the structure of the graph and its edges.

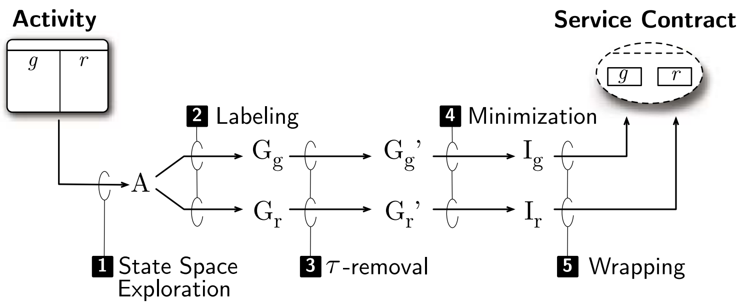
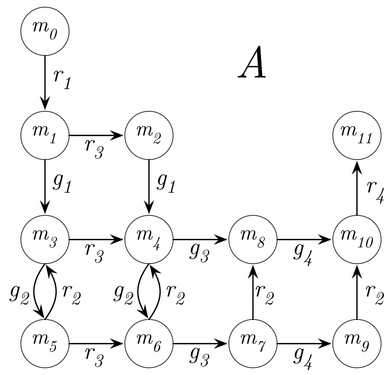


Fig. 7. Overview of the construction mechanism



$$
\begin{aligned}
m_0 &\triangleq i = 1 \wedge t = 0 \wedge s = 0 \wedge a = 0\\
&\quad \wedge q_1 = \langle\rangle \wedge q_2 = \langle\rangle\\
m_1 &\triangleq i = 0 \wedge t = 1 \wedge s = 0 \wedge a = 0\\
&\quad \wedge q_1 = \langle \mathsf{subscribe} \rangle \wedge q_2 = \langle\rangle\\
m_2 &\triangleq i = 0 \wedge t = 0 \wedge s = 0 \wedge a = 0\\
&\quad \wedge q_1 = \langle \mathsf{subscribe}, \mathsf{unsubscribe} \rangle \wedge q_2 = \langle\rangle\\
m_3 &\triangleq i = 0 \wedge t = 1 \wedge s = 1 \wedge a = 0\\
&\quad \wedge q_1 = \langle\rangle \wedge q_2 = \langle\rangle\\
&\vdots
\end{aligned}
$$

Fig. 8. State space for the SMS Notification

4.2 Step 2: Mapping from Actions to Sending and Receiving Labels

The interface graphs that we are generating refer with their edges to the signals sent or received in the activity step the edge represents. Since sending and receiving is relative to the location of the interface (I_r or I_g), we will do a mapping for each of the sides. For that purpose, we look at the definition of the actions in Sect. 3.3 and note which events are visible at the interface. For the interface I_r, the actions map to the following labels:

$r_1 \mapsto\ !\,\mathsf{subscribe}$	$r_2 \mapsto\ ?\,\mathsf{sms}$	$g_1..g_4 \mapsto \tau$
$r_3 \mapsto\ !\,\mathsf{unsubscribe}$	$r_4 \mapsto\ ?\,\mathsf{unsubscribe\_ack}$	

All activity steps executed by the gateway ($g_1..q_4$) do not contribute any visible interaction and thus no label. We therefore map them to the silent step τ. Vice versa, for interface I_g, we read from the activity steps the following mapping:

$g_1 \mapsto\ ?\,\mathsf{subscribe}$	$g_2 \mapsto\ !\,\mathsf{sms}$	$r_1..r_4 \mapsto \tau$
$g_3 \mapsto\ ?\,\mathsf{unsubscribe}$	$g_4 \mapsto\ !\,\mathsf{unsubscribe\_ack}$	

As result we obtain the graphs G_r for the receiver and G_g for the gateway, depicted in Fig. 9.

4.3 Step 3: Removal of τ-Steps

Since interfaces shall describe only observable events, we remove the τ-steps. To do so, we consider each τ-step and do one of the following:

- If the τ-step has the same source and target state, it is simply removed.
- If source and target states of the τ-step are distinct, these states are merged as described below and the τ-step is removed.

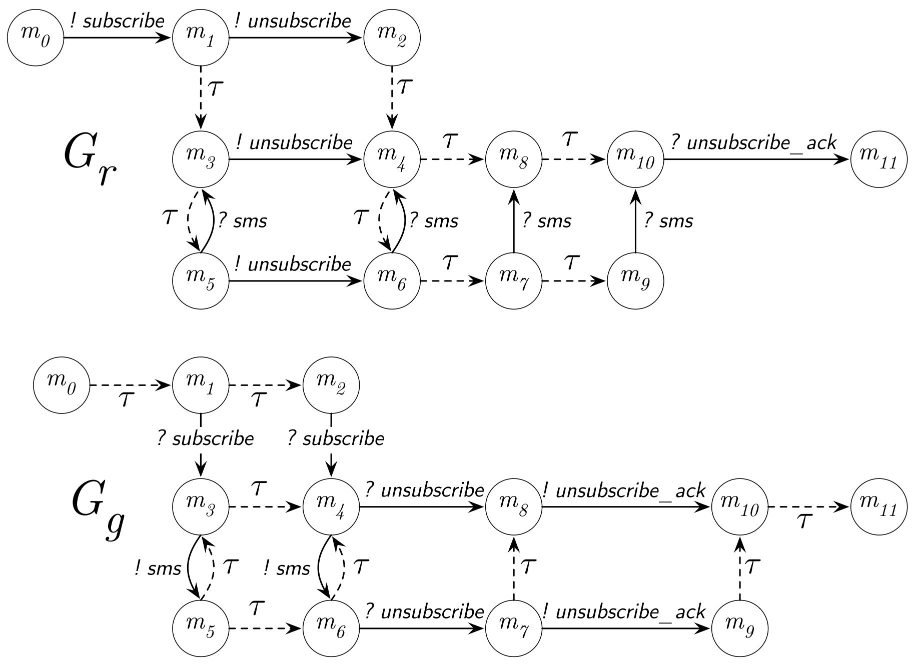


Fig. 9. Mapping to the labels for the receiver (G_r) and the gateway (G_g)

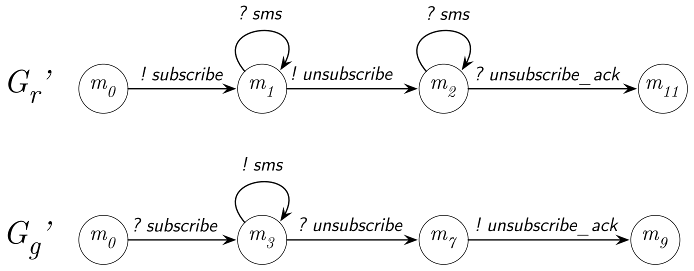


Fig. 10. Graphs after the removal of τ-steps

Merging the source and target state of a τ-step means that all incoming steps into the target state are added to the incoming steps of the source state, and all outgoing steps of the target are set as outgoing steps of the source. The target state is then removed. By subsequently applying removal and merging, several states may be merged into one single state. In the example, states m_1, m_3, and m_5 of G_r are connected by τ-steps, and are therefore merged into state m_1. Similarly, m_2, m_4, m_6, m_7, m_8, m_9, m_{10} are merged into state m_2. States m_0 and m_{11} remain unchanged. As a result, we get the graph G'_r shown in Fig. 10. Similarly, the removal of τ-steps for the receiver yields graph G'_g.

In general, the removal of τ-steps can hide situations in which a component may internally change from a state where a signal can be received to one in which the signal cannot be received anymore (see, for instance, [5,11]). Such τ-steps must not be removed. However, in our case such situations cannot occur in the first place, since the the activity that we use as input for the algorithm describes input consistent behavior, as we will explain in Sect. 5.1. Therefore, all τ-steps may be removed. The merging of states may have the result that between a pair of states there are several steps with the same label. These are duplicates that have no significance for the behavior, and can be merged into one single step.

4.4 Step 4: Minimization

Comparing the graphs of Fig. 10 with the interfaces in Fig. 2, we see that our algorithm has already produced the desired result. In general, however, the graph obtained after the removal of all τ-steps can contain equivalent states, i.e., states from which the same behavior (defined by the possible sequence of signal transmissions) can be observed.

To reduce the interfaces to their minimal representation, we use the minimization procedure for finite state machines described by Holzmann in [13]. This procedure checks if a pair of states is equivalent by considering if (1) all possible labels of a state are also enabled by the other, and (2), if all possible successor states of one state are matched by at at least one possible successor of the other state. The second condition is evaluated iteratively, using an equivalence matrix. Once the algorithm terminates, groups of equivalent states can be read from the equivalence matrix. These states are merged, similar to the merging during the removal of τ-steps. Since this may again result in multiple steps with the same label between a pair of states, we remove duplicate steps during merging.

4.5 Step 5: Encapsulation as Service Contracts

From the graphs obtained after minimization we construct UML state machines as the final representation for the service contracts. These state machines are stored within a UML model repository, for which we use the one provided by the Eclipse Modeling Project [9]. Each state of a graph is represented by a UML state. Graph states with no outgoing edge are represented by UML final states, and the initial state is marked by an initial transition originating from an initial pseudo state. The edges are represented by UML transitions. Receiving labels (?) are translated to corresponding signal triggers, and sending labels (!) to send signal actions as part of a transition's effect. The signals referred by these sending actions and triggers declare typed attributes if the activity flows they represent transport objects, so that the interface also defines which data types are required.

5 Discussion

5.1 Soundness of the Generated Interfaces

By taking activities as the starting point, we can utilize a property of activities that distinguishes them from the state machines used to model interfaces: State machines must for every control state explicitly declare signals they can receive. We recall the example from the introduction, in which *s_r3* in Fig. 2 must explicitly declare *?sms*. Forgetting the declaration of this reception results in input inconsistent behavior, and may result in unspecified receptions.

In contrast, activities do not exhibit unspecified receptions: Signal sendings and receptions are modeled by the very same element, a single flow crossing partition borders. Whenever a token representing a signal is at the head of a queue entering a partition, it may continue and trigger the behavior declared by the downstream flow.[3] The activities we take as input for the generation of interfaces therefore describe input-consistent behavior.

To ensure, that the interfaces generated by the algorithm do in fact describe the behavior observed in the activity, we reason in temporal logic. The interfaces need to be abstractions of the complete behavior described by the activity, or, vice-versa, the activity must be a valid refinement of all the interfaces. In temporal logic, this can be verified by a refinement proof. Both, the complete behavior of the collaboration A and the generated interfaces I_i, can be represented by TLA specifications S_a and S_i, respectively. In TLA, refinement corresponds to implication ($\Rightarrow$). This means we have to prove that for all interfaces i, $S_a \Rightarrow S_i$ holds. With refinement mappings, TLA offers a quite elegant method to perform this implication proof (see [1]). For that, we have to find a mapping R from the state spaces of S_a to the one of S_i, so that:

– The initial state of S_a maps to the initial state of S_i.
– An action of S_a, leading the system specification from state s to t, maps to either an action in S_i so that $R(s)$ leads to $R(t)$, or to a stuttering step, i.e., the values of S_i stay unchanged.

Since our algorithm takes the more detailed system S_a as input and constructs the interfaces through abstraction, it is always possible to give the refinement mapping R by observing which states are merged during the τ-removal and the minimization. For the interface I_r, we obtain therefore the following mapping for the state variable *state* of the interface I_r:

$$\begin{aligned} \mathit{state} = {} & \text{IF } m_0 \text{ THEN } \mathit{sr\_1} \\ & \text{ELSE IF } m_1 \vee m_3 \vee m_5 \text{ THEN } \mathit{sr\_2} \\ & \text{ELSE IF } m_2, m_4, m_6, m_7, m_8, m_9, m_{10} \text{ THEN } \mathit{sr\_3} \\ & \text{ELSE } \mathit{sr\_4} \end{aligned}$$

[3] If this behavior would harm any of the properties in Sect. 3.4, it would be detected by the analysis and the construction of the interface would be aborted.

We have verified that this refinement mapping is correct, i.e., $S_a \Rightarrow S_i$, using TLC [30]. Due to this refinement relation between an activity and the interfaces generated from it, as well as the reception behavior of activities explained above, the generated interfaces are sound.

– Property (i) is fulfilled since an interface is generated from the state space that includes all possible signal receptions. For each state in which a signal *sig* is at the head of the incoming queue for interface I, the algorithm adds an outgoing transition *?sig* to the current state of I. Since the only action removing *sig* from the head of the queue is the actual reception by I, all states in which *sig* may be received will have *?sig* as an outgoing transition for I. Hence, an interface declares receiving operations whenever they are required. This also implies that interfaces are input consistent, since all signal sendings lead to corresponding transitions in the interface.
– Property (ii) is fulfilled due to the requirement that activities are free from deadlocks, ensured by the prior analysis.

5.2 Scalability of the Method

Although we explore all reachable states of a specification, our method is scalable, due to the compositional semantics of the underlying engineering method introduced in Sect. 3.1.

– The collaborations, for which we generate the contracts, are typically related to a specific task, so that they are limited in complexity to begin with.
– If a collaboration needs to interact with other functionalities, these may be abstracted by internal signals (like *s:SMS* and *a:ACK* in Fig. 4) which separates our collaboration from the gateway's logic. Another possibility is to describe expected behavior on either collaboration side by ESMs as mentioned in Sect. 3.1. In both cases, the state space is limited to the functionality needed to realize the collaboration's behavior.
– If an activity is composed of other building blocks on either side, these would be in turn abstracted by their ESMs.

5.3 Related Approaches

Several other approaches in literature deal with the compatibility of interfaces. In [11], Floch describes a process that extracts interfaces (called *association roles*) from SDL processes to document the observable behavior towards each communication partner. This facilitates the validation of systems under design, since state machines under construction can be validated against such interfaces. However, within these processes designers still need to deal with the concepts of interfaces and their intricacies as discussed in the introduction, and may lose focus on the actual application. Sanders extends this approach in [26] with progress labels. These labels can be used when a service should be chosen at run-time: Instead of just selecting any service that is compatible, the service that enables the most features can be chosen. In SIMS [28], also sub-typing of interfaces and ontologies [23] were discussed in that context.

Differences of our work lie especially in the implied development method: we propose the use of behaviorally complete defined collaborations in terms of activities as the starting point, from which the service contract and necessary interfaces are derived. Since we focus on pure interface compatibility in terms of allowed signal sequences, we have not treated progress or ontologies. These can be seen as additional criteria that are meaningful during service discovery, in case that we can choose among several compatible services.

6 Concluding Remarks

We presented an automated process for the synthesis of compatible interfaces and service contracts, starting with behaviorally complete and application-oriented descriptions expressed by UML activities. The synthesis algorithm is implemented as an Eclipse plug-in in our tool suite Arctis [2], and has been tested on numerous collaborations from other case studies. Typically, the algorithm produces the interfaces within the fraction of a second, due to the generally small state space of collaborative descriptions.

The novelty of our method lies in its practical implications on development. When creating a system that should make some of its functionality available via service contracts, engineers can focus on their application expressed by activities, and derive corresponding service contracts automatically. This not only saves time, but also avoids a manual process which can lead to inconsistencies.

For an inverted scenario (that is, when a designer should build a component that corresponds to a *given* service contract) we can generate special building blocks from service contracts that guarantee compliance, and any violations will be detected by our analysis tools. We have demonstrated and implemented such an approach in [20] for interfaces given as web service descriptions. In either scenario, our method does not require developers to understand the intricacies of behavioral interfaces, which in our opinion is a major hurdle for their widespread adoption.

References

1. Abadi, M., Lamport, L.: The Existence of Refinement Mappings. Theoretical Computer Science 82(2), 253–284 (1991)
2. Arctis Website, http://arctis.item.ntnu.no
3. Back, R.J.R., Kurki-Suonio, F.: Distributed Cooperation With Action Systems. ACM Transactions on Programming Languages and Systems 10(4), 513–554 (1988)
4. Ben-Abdallah, H., Leue, S.: Syntactic Detection of Process Divergence and Non-Local Choice in Message Sequence Charts. In: Brinksma, E. (ed.) TACAS 1997. LNCS, vol. 1217, pp. 259–274. Springer, Heidelberg (1997)
5. Bræk, R., Haugen, Ø.: Engineering Real Time Systems: An Object-Oriented Methodology Using SDL. The BCS Practitioner Series. Prentice-Hall, Englewood Cliffs (1993)

6. Broy, M., Krüger, I.: Interaction Interfaces - Towards a Scientific Foundation of a Methodological usage of Message Sequence Charts. In: ICFEM 1998: Proceedings of the Second IEEE International Conference on Formal Engineering Methods, p. 2. IEEE Computer Society, Washington (1998)
7. Carrez, C., Fantechi, A., Najm, E.: Behavioural Contracts for a Sound Assembly of Components. In: König, H., Heiner, M., Wolisz, A. (eds.) FORTE 2003. LNCS, vol. 2767, pp. 111–126. Springer, Heidelberg (2003)
8. de Alfaro, L., Henzinger, T.A.: Interface Automata. SIGSOFT Softw. Eng. Notes 26(5), 109–120 (2001)
9. Eclipse Modeling Project, http://www.eclipse.org/modeling
10. Engelhardtsen, F., Prinz, A.: Application of Stuck-free Conformance to Service-Role Composition. In: Gotzhein, R., Reed, R. (eds.) SAM 2006. LNCS, vol. 4320, pp. 115–132. Springer, Heidelberg (2006)
11. Floch, J.: Towards Plug-and-Play Services: Design and Validation using Roles. PhD thesis, Norwegian University of Science and Technology, Trondheim (2003)
12. Herrmann, P., Krumm, H.: A Framework for Modeling Transfer Protocols. Computer Networks 34(2), 317–337 (2000)
13. Holzmann, G.J.: Design and Validation of Computer Protocols. Prentice Hall Software Series. Prentice-Hall, Englewood Cliffs (1991)
14. Kraemer, F.A.: Engineering Reactive Systems: A Compositional and Model-Driven Method Based on Collaborative Building Blocks. PhD thesis, Norwegian University of Science and Technology, Trondheim (2008)
15. Kraemer, F.A., Bræk, R., Herrmann, P.: Compositional Service Engineering with Arctis. Telektronikk, vol. 1.2009 (2009)
16. Kraemer, F.A., Herrmann, P.: Service Specification by Composition of Collaborations — An Example. In: WI-IATW 2006, pp. 129–133. IEEE Computer Society, Washington (2006)
17. Kraemer, F.A., Bræk, R., Herrmann, P.: Formalizing Collaboration-Oriented Service Specifications using Temporal Logic. In: Networking and Electronic Commerce Research Conference 2007 (NAEC 2007), pp. 194–220. ATSMA Inc., USA (2007)
18. Kraemer, F.A., Herrmann, P.: Transforming Collaborative Service Specifications into Efficiently Executable State Machines. In: Ehring, K., Giese, H. (eds.) Proceedings of the 6th International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2007). Electronic Communications of the EASST, vol. 7 (2007)
19. Kraemer, F.A., Herrmann, P., Bræk, R.: Aligning UML 2.0 State Machines and Temporal Logic for the Efficient Execution of Services. In: Meersman, R., Tari, Z. (eds.) CoopIS, DOA, GADA, and ODBASE 2006. LNCS, vol. 4276, pp. 1613–1632. Springer, Heidelberg (2006)
20. Kraemer, F.A., Samset, H., Bræk, R.: An Automated Method for Web Service Orchestration based on Reusable Building Blocks. In: Proceedings of the 7th International IEEE Conference on Web Services (ICWS) (to be published, 2009)
21. Lamport, L.: Specifying Systems. Addison-Wesley, Reading (2002)
22. Mencl, V.: Specifying Component Behavior with Port State Machines. Electr. Notes. Theor. Comput. Sci. 101, 129–153 (2004)
23. Michał, R., Meland, P.-H., Floch, J., Domaszewicz, J.: Ontology-based Use Cases for Design-time and Runtime Composition of Mobile Services. In: Proceedings of the Mobile Data Management Workshops MDMW 2008, pp. 29–36. IEEE Computer Society Press, Washington (2009)

24. Object Management Group: Service Oriented Architecture Modeling Language (SoaML) - Specification for the UML Profile and Metamodel for Services (UPMS), version 2.0 (November 2008), http://www.omg.org/docs/ad/08-11-01.pdf
25. Samset, H., Bræk, R.: Dealing with Active and Stateful Services in the Service-Oriented Architecture. In: Di Nitto, E., Ripeanu, M. (eds.) ICSOC 2007. LNCS, vol. 4907, pp. 268–281. Springer, Heidelberg (2009)
26. Sanders, R.: Collaborations, Semantic Interfaces and Service Goals: A Way Forward for Service Engineering. PhD thesis, Norwegian University of Science and Technology, Trondheim (2007)
27. Sanders, R., Bræk, R., von Bochmann, G., Amyot, D.: Service Discovery and Component Reuse with Semantic Interfaces. In: Prinz, A., Reed, R., Reed, J. (eds.) SDL 2005. LNCS, vol. 3530, pp. 85–102. Springer, Heidelberg (2005)
28. SIMS Project Website, http://www.ist-sims.org
29. Slåtten, V.: Automatic Detection and Correction of Flaws in Service Specifications. Master's thesis, Norwegian University of Science and Technology, Trondheim (2008)
30. Yu, Y., Manolios, P., Lamport, L.: Model Checking TLA$^+$ Specifications. In: Pierre, L., Kropf, T. (eds.) CHARME 1999. LNCS, vol. 1703, pp. 54–66. Springer, Heidelberg (1999)

The Challenges of Using SDL for the Development of Wireless Sensor Networks

Klaus Ahrens, Ingmar Eveslage, Joachim Fischer,
Frank Kühnlenz, and Dorian Weber

Humboldt-Universität zu Berlin, Department of Computer Science,
Unter den Linden 6, 10099 Berlin, Germany
{ahrens,eveslage,fischer,kuehnlenz,weber}@informatik.hu-berlin.de

Abstract. In recent years, Wireless Sensor Networks (WSNs) have been primarily used to build ad-hoc telecommunication infrastructures from scratch or as low-cost alternatives to traditional networks. But the diversity of applications with typically narrow node resources and requirements of already existing information infrastructures sets hard constraints to WSN. The software development process becomes even more complicated when real-time constraints have to be taken into account. This is the case when the physical processes of the WSN environment have to be observed and are realized in space and time. For the development of such WSN we present a model-based framework (GAF4WSN), where the well-known techniques SDL, UML and ASN.1 are involved. The framework was already successfully used for the development of a new generation of Earthquake Early Warning Systems (EEWS). An Earthquake Synthesizer (ES) and an Experiment Management System (EMS) complete the framework, which supports the modelling, simulation, installation and administration of different EEWS approaches in combination with a Geographic Information System (GIS).

Keywords: model-based development, sensor systems, wireless sensor networks, simulation, code generation, experiment management, SDL, UML, ASN.1.

1 Introduction

Wireless Sensor Networks (WSN) become more and more popular for monitoring numerous physical phenomena and often they should be self-organized for easy installation and maintenance (e.g. to autonomously integrate new sensor nodes and react on failed ones). This emerging technology offers exciting potential for numerous application areas including environmental, medical, military, transportation and disaster management. The major challenges in the WSN domain include sensing and collecting data from sensors and then evaluating it to formulate meaningful information such as generating alarms or supporting decisions while minimizing energy consumption. But the development of such complex systems is a challenging task. In particular, the evaluation of their potential real-time behaviour is almost impossible or too expensive without prior modelling experiments, involving computer simulations. Besides the complexity of the system itself, its surrounding environment in real-time behaviour terms must be considered and modelled (e.g. load-models using synthetic sensor input data). In addition to this kind of requirement, it is quite difficult to port a WSN application to different platforms.

To solve these issues, system engineers need to be able to model applications using high-level abstractions and to simulate those using configurable and realistic topologies for the network itself. We follow a generally approved model-driven development paradigm using a technology mix of SDL/ASN.1/UML/C++ [1,2] to generate the code for the target hardware platform (sensor nodes) and for different kinds of simulators supporting different experiment scenarios (including the system and its environment) in preparation for the implementation. A further specialty of our approach is the integration of the model-driven tool chain into a spatial-time-based Experiment Management System (EMS) in connection with a Geographic Information System (GIS). This allows us to describe the WSN topology and the distribution or movement of the physical phenomena in a geographic map. All tool components are integrated by our GIS-based Development and Administration Framework for Wireless Sensor Networks (GAF4WSN).

The presented paper illustrates the usage of GAF4WSN developing a new approach for Earthquake Early Warning Systems (EEWS) that uses self-organizing mesh WSN based on commercial off-the-shelf hardware where the WSN nodes are equipped with low-cost seismometers and GPS units. The most important installed software components are the Linux operating system, WLAN network stack and further protocol units for message routing and alarming through a meshed network. Additional general services are provided by middleware between the communication and application layer of the protocol stack. Significant and innovative aspects of that approach are connected with the fact that each sensing node performs on-site independent analysis of the ground motion and that the early warning is automatically carried out within the wireless mesh network itself by dedicated alarming processes. Moreover, since commercial off-the-shelf hardware is inexpensive, this also allows for more sensor nodes and hence much denser sensor networks. These can provide more detailed, higher resolution information than traditional seismic networks with only a few powerful seismological stations spread over a large area.

Structure and behaviour models of network topologies developed with GAF4WSN for specific geographic regions are coupled with seismometer sensor input data generation and convenient visualizations to form the basis for various types of simulation experiments ahead of system implementation and installation. The general objective of these studies is to test the functionality of an EEWS and to optimize it with respect to real-time, reliability and cost requirements of potential end-users. This approach is used for implementing a prototype-EEWS developed within the EU project SAFER (Seismic eArly warning For EuRope, [3]) in cooperation with the GeoForschungsZentrum Potsdam. A small installation of this prototype system is established in the mega-city Istanbul, a region threatened by strong earthquakes.

This paper focuses on a transcompiler (integrated in GAF4WSN) as an extension of PragmaDevs RealTimeDeveloperStudio [4] to generate specific code for different types of simulation experiments and for the sensor nodes' target platform out of SDL/UML models.

2 Related Work

Legacy computer network simulators, such as ns-2 [5], ns-3 [6], JistSwans [7] and OPNET [8] enable the simulation of wired or wireless network behaviour and protocol stack operation, but do not take into account WSN characteristics. This is overcome in the simulators proposed specifically for WSN, which were categorized by [9] into networking oriented and sensor node simulators. In addition [9] introduces a further simulator type, which is characterized by an integration of design, simulation, debugging and code generation tools under a unique GUI.

The **network-oriented simulators** model the transmission medium in detail and are more suitable for the large scale WSN simulations. Most of the proposed networking oriented simulators are based on legacy computer network simulators. SensorSim [10] and Naval Research Laboratory's sensor network simulator [11] extend ns-2 with general WSN features.

The **sensor node simulators** mainly simulate the operation of a single node but implement a lightweight communication model. Most of the proposed sensor node simulators are targeted to TinyOS motes. Complete TinyOS systems can be simulated with TinyOS SIMulator (TOSSIM) [12], and TinyOS Scalable Simulation Framework that is an extension to SWAN [13].

Currently some **integrated tool environments** are available that combine graphical design techniques with different kinds of simulators and code generators for different platforms. It is interesting that the most used modelling technique in that area is the ITU-T language SDL. Examples of that category are the WISENES framework [9] or the complete framework for modelling, simulation and multiplatform code generation based on MathWorks tools. They enable system engineers to effectively design WSN applications and map them on a wireless network. Application developers can automatically generate the complete application code for several target operating systems from the same simulated and debugged model, without thinking about the details of the target platform implementation. The SDL Environment Framework (SEnF), developed by the group of R. Gotzhein [14], is a further prominent example of that category.

Our integrated tool environment **GAF4WSN** supports time-dependent and timeless simulations in combination with virtual or real machine simulations based on a sensor network model in SDL/UML. Furthermore it is possible to use a GIS-based approach for modeling different net topologies in combination with models of environmental processes. For efficient transmission of messages, ASN.1 coding/decoding mechanisms are offered. GAF4WSN is not only a framework for the design, simulation and code generation; furthermore it is also prototyped as an operating and management platform for WSN. For that the WSN have to be equipped with a middleware in combination with an EMS, which has to be installed outside the WSN.

3 SOSEWIN: A Wireless Meshed Seismic Sensor Network

The Self-Organizing Seismic Early Warning Information Network (SOSEWIN) is technically a decentralised, wireless mesh sensor network, made up of low-cost components, with a special seismological application that supports earthquake early warning and rapid response tasks. It is being developed within the SAFER [15] project funded by the European Union.

3.1 Earthquake Early Warning

Earthquakes belong to the most devastating natural hazards. They not only cause damage to economic infrastructures, but also cause the loss of human life. Several mega-cities like San Francisco, Mexico City, Tokyo or Istanbul are at risk. Such cities not only accommodate a large number of people, but they also constitute the economic heart of their regions. An EEWS is feasible because earthquakes generate two basic kinds of seismic waves: P-waves (primary waves) and S-waves (secondary waves). The harmless P-waves are almost twice as fast as the S-waves, which cause most of the destructive shaking. Therefore, the warning time (time interval between the detection of the fast P-waves and the arrival of the slower S-waves), depends on the distance of a target area, usually a city to be protected, from the hypocenter (the origin location of the earthquake). Although for the worst case scenario this short warning time is not enough for people to leave their houses, this can still be sufficient to mitigate secondary damages like gas explosions and fire outbreaks.

3.2 Rapid Response

Immediately after an earthquake event, several analyzing tasks of the recorded seismic waves must be carried out in order to understand the seismic impacts of the event. These analyzing tasks belong to the rapid response phase. A typical task in this phase is the fast generation of so-called ShakeMaps [16], which show the wave peaks (or intensity) in the seismic affected area in form of isobar lines or different colours. The combination of such ShakeMaps with information about building structures and population densities in the affected area is important for fast and proper disaster management. Thus the recording of aftershocks directly in the affected area is a typical task or earthquake task forces.

3.3 Characteristics and Purpose

In contrast to existing EEWS [17], SOSEWIN realizes the new approach of performing the decision whether an earthquake early warning should be raised, inside the network itself and not in a central data centre (two-level alarming protocol, Sect. 6.1).

A SOSEWIN reacts in a self-organizing way on newly added or removed nodes. Therefore it is more robust, easier to install and maintain than traditional, centralized EEWSs. In addition, with a relatively low price compared to common seismometer stations, a very dense network (hundreds or thousands of nodes) can be established directly in the threatened region. SOSEWIN nodes are also small and light-weight, which allows a working network to be established very quickly to support earthquake task forces during the rapid response phase.

The SOSEWIN nodes were already successfully used in two field tests for monitoring building structures: Recently during the aftershock phase of the Abruzzo event (Italy) and the Fatih bridge (Istanbul, Turkey) [18].

4 Framework Requirements for the Design and Administration of WSN

In the following section we postulate a list of requirements which we were able to identify by generalization of the SOSEWIN development in our case study and which were introduced step by step in our framework GAF4WSN as special features.

- To test SOSEWIN in the form of models or real installed network topologies, a framework feature is necessary as it allows us to simulate the **environmental processes** (characterized by time series for each node, identified by its precise local geographic position). In the case of earthquakes this data has to be generated by a synthesizer tool for which input data is defined by an event description (epicentre, depth, magnitude), a geological description of the ground area of the travelling waves (defined by fault characteristics), the WSN topology (given by geographic positions and link quality parameters) and the node equipment (processor, memory, GPS-unit, seismometer sensor, software components). Independent of the concrete use case, the framework should support the built-up of model repositories. Furthermore the WSN middleware should support a deployment of the test data and a principle switch between two kinds of node input: real sensor data or synthesized data.
- For the configuration of WSN models or real prototypes (network topology, software architecture of nodes, geographic area) influenced by environmental processes, a **graphical Topology Editor** based on a **Geographic Information System (GIS)** is necessary. Adding and removing nodes is one of the necessary basic features. The OGC standard Web Feature Service (WFS) [19] can be recommended to introduce an additional layer of spatial objects (e.g. points and lines with additional attributes) in an existing topographic map.
- The example SOSEWIN has shown that for the design of WSN as alarming systems, **time-based simulations** are absolutely essential to estimate the relation between the speeds of the environmental process (travelling of seismic waves) and the information processing process by the WSN. This kind of performance prediction experiments allows for the evaluation of the chosen network topology, routing protocols, message coding methods and hierarchical alarming protocols before the WSN is installed as an early warning system.

- In view of the complexity of large-scale WSN, the spread of WSN simulators should allow a **step-wise transition of abstract to real components** during the test process of WSN components. Here it was very useful to offer the choice between different time concepts (free model time, model time related to real time and real time itself). Furthermore tests should be supported by execution of complete WSN protocol stacks on virtual machines, even if only small net topologies can be investigated by that approach. Larger topologies need a higher degree of model abstraction, so that GAF4WSN should support the use of ordinary network simulators.
- The WSN application protocols should be designed by **standardized modelling techniques**. This has an impact for maintaining and porting the WSN concepts. As already mentioned in Sect. 2, SDL was also the favourite modelling language for our framework. We decided to use the Real-time Developer Studio from PragmaDev, which supports the design of real-time systems combined with debugging and C code generation facilities for different real-time operating systems. In particular this tool allows for a powerful combination of the SDL agent concept (except some restrictions) with UML class, use case and sequence diagrams. The data and sequential action parts of PragmaDevs SDL/UML harmonization are based on C/C++. As a consequence pointers can be efficiently handled as message parameters independent of whether the message is used for internal node communication (with the parameter as a reference value) or if it is used for node-to-node communication (with the parameter as a dereferenced value).
  This piece of knowledge, whether the parameter is a reference or a dereferenced value, is always given by the respective sender agent. In addition it is possible to bypass the actual weakness of SDL-RT in message delivery between processes of block instances, where in SDL-RT all instances of a block share process instance sets: one shared instance set for each enclosed process definition (compared with SDL-2000 where each process instance set in a block is enclosed by the block and distinct from sets in other block instances).
- Commonly WSN have to deliver a great quantity of raw sensor data. Standardized techniques for coding and decoding improve the efficiency of the whole network. We use **ASN.1 basic encoding/decoding** rules with Mini-SEED [20], a standardized coding technique for seismometer raw data.
- Experiments as generalization of systematic simulator runs or real tests of WSN prototypes form the evaluation base of our use case example SOSEWIN. To ensure the consistency of all artefacts which are necessary for an experiment (concerning WSN model, WSN environment, tools, test input and output results), a tool-based information management is absolutely convenient. Such an experiment management system should be integrated in the design and administration framework. Independent of the nature of the investigation object (model or real prototype), all events like state changes, timer events or message arrivals have to be logged into files. This guarantees a unique operational interface for the **experiment management system**. The collected data form the base for a further use-case-dependent information evaluation.

- Equally important are issues related to network deployment, interworking and the integration with existing network infrastructures. Of particular interest here is the choice of a suitable **communication middleware** that hides the complexity of the low-level networking protocols and facilitates the development of applications. In this context the data-centric communication approach, which is based on data content rather than receivers' addresses, appears to be more efficient and more appropriate for wireless sensor networks. This middleware can act as "glue" between software components of applications and the network or it is the "slash" in the Client/Server construct. To act in this way, middleware should offer services specified by well-defined interfaces.

5 Our Approach: GAF4WSN

5.1 Architecture Overview

Figure 1 shows an overview on the core components of GAF4WSN. For supporting the development task, this framework offers a centralized management of models, software artefacts and experiment results. This is enabled by several repositories that are implemented using database technologies with spatial extension (OpenGIS). This GIS-based technology also offers the possibility to administrate a WSN in terms of monitoring its health status or for deploying new or updated software components. The following gives a brief description of the framework components shown in Fig. 1.

- The **Experiment Management System** supports planning, configuration, automated execution of experiments and storage of experiment results. It provides additionally GIS-based visualization capabilities for experiment results and it can also be used for planning software deployment and monitoring of an installed WSN.
- The **Model Repository** stores used SDL(-RT), UML and C++ models defining the application for the WSN and underlying layers like middleware [21] or routing.
- The **Model Configurator** knows the target platform and uses platform dependent artefacts to configure the compiler (e.g. cross-compilation). It also specifies certain input parameters (e.g. threshold values or network clustering) and stores the whole configuration into the Experiment Repository.
- The **Transcompiler** is indeed a tool chain of several transcompilers, which accept SDL-RT models and compile C++ code at the end into different executable binaries (simulators, target code). See Sect. 5.3 for details.
- The **Execution Libraries** are used by the transcompiler to generate the executable binaries for a simulator or target code. They comprise several libraries for the target code binary (e.g. threading and networking capabilities with the necessary functions to decode and encode the network messages using ASN.1). For simulator support it is also a pool of simulation frameworks.

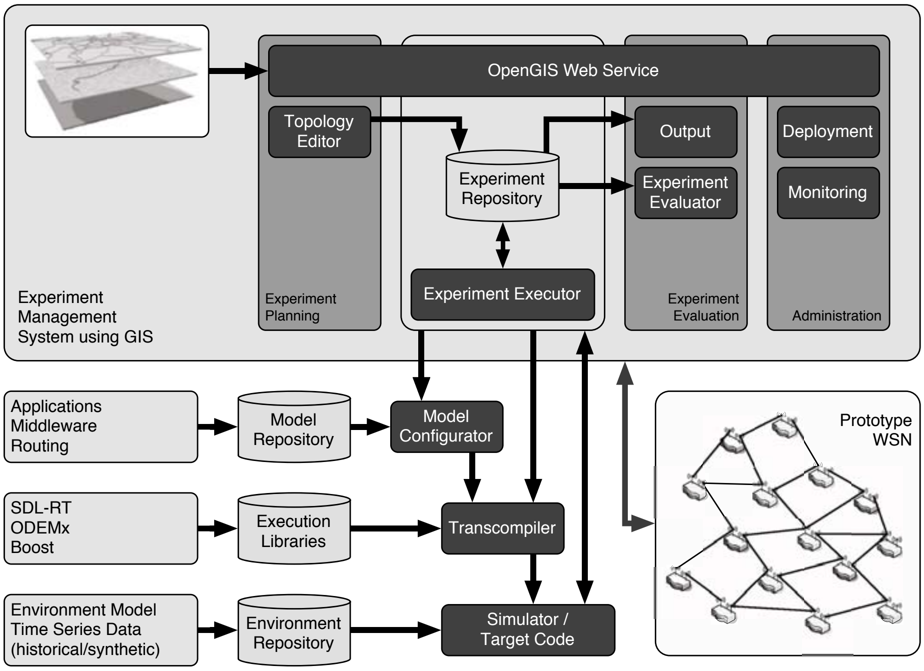


Fig. 1. Overview of GAF4WSN architecture

- The **Environment Repository** comprises environment models to describe the setting of the WSN (e.g. network topology, link qualities) for simulation and also load models to provide synthetically or historically recorded sensor data to be applied as input to a sensor node in order to execute certain experiments (simulative or with an existing WSN).
- The **Simulator / Target Code** is the target binary representation created by the transcompiler tool chain. It is either a simulator out of a set of several simulator types realizing different functionality (derived from different simulation frameworks) or a binary intended to run on a node of a WSN.

Further on in this paper we focus on certain components of the GAF4WSN: mainly the transcompiler tool chain (Sect. 5.3) addressing the special needs to map SDL modelling concepts to simulator and target code. But in the following we first introduce the basic libraries for these target platforms.

5.2 Used Libraries / Target Platform

ODEMx. [22] is a general purpose simulation C++ library that follows the well-known Simula-67 tradition of process-oriented model descriptions with model time based interleaved execution. ODEMx uses a highly portable coroutine implementation and has been developed by the authors in several versions along with the emergence of C++ features. ODEMx provides base classes for processes and events, so that it supports closed process life cycles as well as simple event actions. Processes are scheduled according to their model time consumption and executed as coroutines. Events are executed by simple function calls on top of some process' execution stack. The library contains special classes for synchronization utilities as limited resources and unlimited queues. It provides a broad range of random distribution classes and automatic reports on all resource utilizations. ODEMx also supports combined models with both discrete and continuous time processes. Since its version 2.0 ODEMx provides a special package for modelling protocol components. Using ODEMx allows for highly adaptable experiments with the signalling protocols specified in SDL with different variation points:

- various levels of details in the model representation of the net infrastructure ranging from ideal transmission without signal loss to tuning of transmission quality parameters including group separation and explicitly modelling of lower protocol layers,
- inclusion of real world node setup scenarios as well as all kinds of experiments on hypothetic node allocations,
- feeding synthesized (but real world equivalent shaped) or recorded real sensor data into the sensor nodes.

Boost. [23] is a collection of C++ libraries usable across a broad spectrum of applications. It aims at establishing "existing practice" and provides reference implementations so that Boost libraries are suitable for eventual standardization. For the target platforms, the following particular components are used for the target code generation:

portable networking — including sockets;
timers;
hostname resolution;
socket iostreams;,
portable C++ multi-threading.

OpenWrt. [24] is described as a Linux distribution for embedded devices. Instead of trying to create a single, static firmware, OpenWrt provides a fully writable file system with package management. This frees us from the application selection and configuration provided by the vendor and allows us to customize the device through the use of packages to suit any application. For the developer, OpenWrt is the framework to build an application without having to build the complete firmware around it.

5.3 SDL HUB Transcompiler

General Description. The core tool of our framework GAF4WSN is the extension of PragmaDevs RTDS to a new transcompiler which is able to generate C++ code artefacts from UML/SDL-RT. These different artefacts can be linked after their compilation with different libraries, which allows the construction of different binaries corresponding to Fig. 2:

A) RTDS-Simulator of small WSN topologies (as direct SDL interpretation)
B) ODEMx-Simulator of large-scale topology applications without a precise model of the underlying communication layer
C) OpenWrt-target code fragments representing the application functionality which has to be installed together with other software components on a node of a special type, whose functionality is described in SDL by a corresponding SDL block type and executable under OpenWrt.

In addition an SDL transcompiler to ns-3 [6] is in development by us.

PragmaDev approach. As illustrated in Fig. 2.A the PragmaDev approach is given by translation of correct SDL-RT models into valid C code in general. Some parts of the output are generated; others are static in order to support a wide range of operating systems. The static parts are interleaved by the RTDS transcompiler with generated code artefacts, which are parameterized by using macro functions. Each framework provided by PragmaDev or any third party developer both needs to define the macros used by the code generator as well as the static parts. These components are represented by "RTDS code templates".

During the transformation into C code, RTDS discards any information connected to the composition of processes into blocks. This implies a flat hierarchy among blocks and processes. Processes are always mapped to threads of the corresponding OS, procedures map to functions and messages are represented as C structures.

Due to the lack of a representation of blocks in the generated source code, RTDS is not able to instantiate blocks dynamically. This motivates the use of an additional compiler that gets called when the PragmaDev one finishes its work.

RTDS extension points. Figure 2.A also illustrates the principle way of compiler extension. First of all definitions of generated macro calls can be changed using customized RTDS code templates. In addition, the static files that are interleaved with actual generated code can be used as "markers" for our tool during examination of the generated files. The advantage of this approach is that the customised tool doesn't need to really understand the structure of the generated code, but instead can use the markings for orientation. Finally RTDS allows to partially redefine generated makefiles by the use of (RTDS generated) include directives. This mechanism can be used to integrate additional tools seamlessly into the chain without artificially changing how the toolchain works.

HUB SDL approach. Due to the structure of the generated code, our additional transcompiler is still able to reconstruct missing information. Blocks are mapped to C++ classes that are populated with other blocks or processes, and processes become C++ classes that have procedures as their methods. In order to allow the sending of messages among different processes, all processes derive from a common base class that implements an interface for resolving process type IDs. Since we require dynamic instantiation of blocks in our large-scale

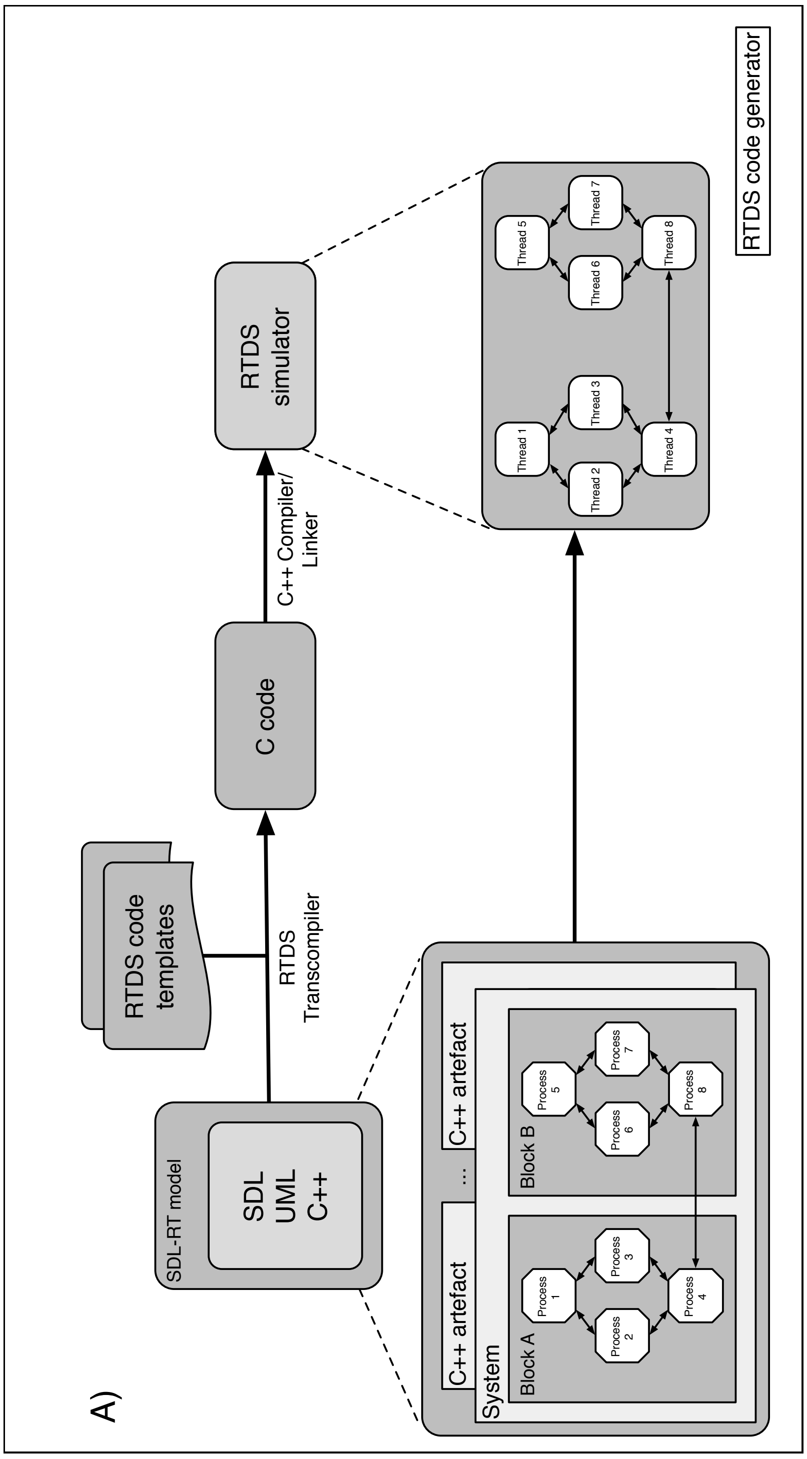


Fig. 2. A. Different code generation target types — RTDS-Simulator

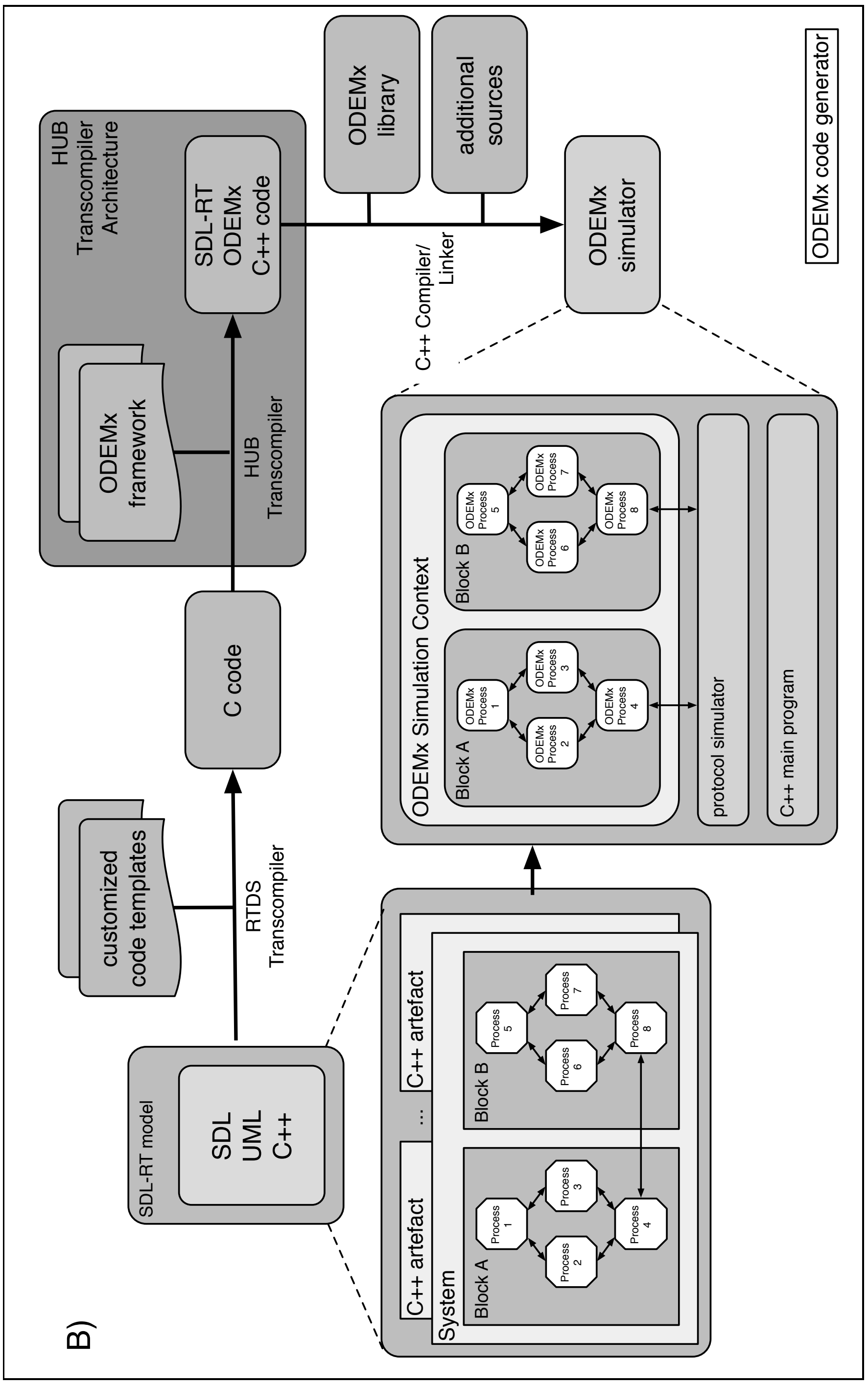


Fig. 2. B. Different code generation target types — ODEMx simulator generation

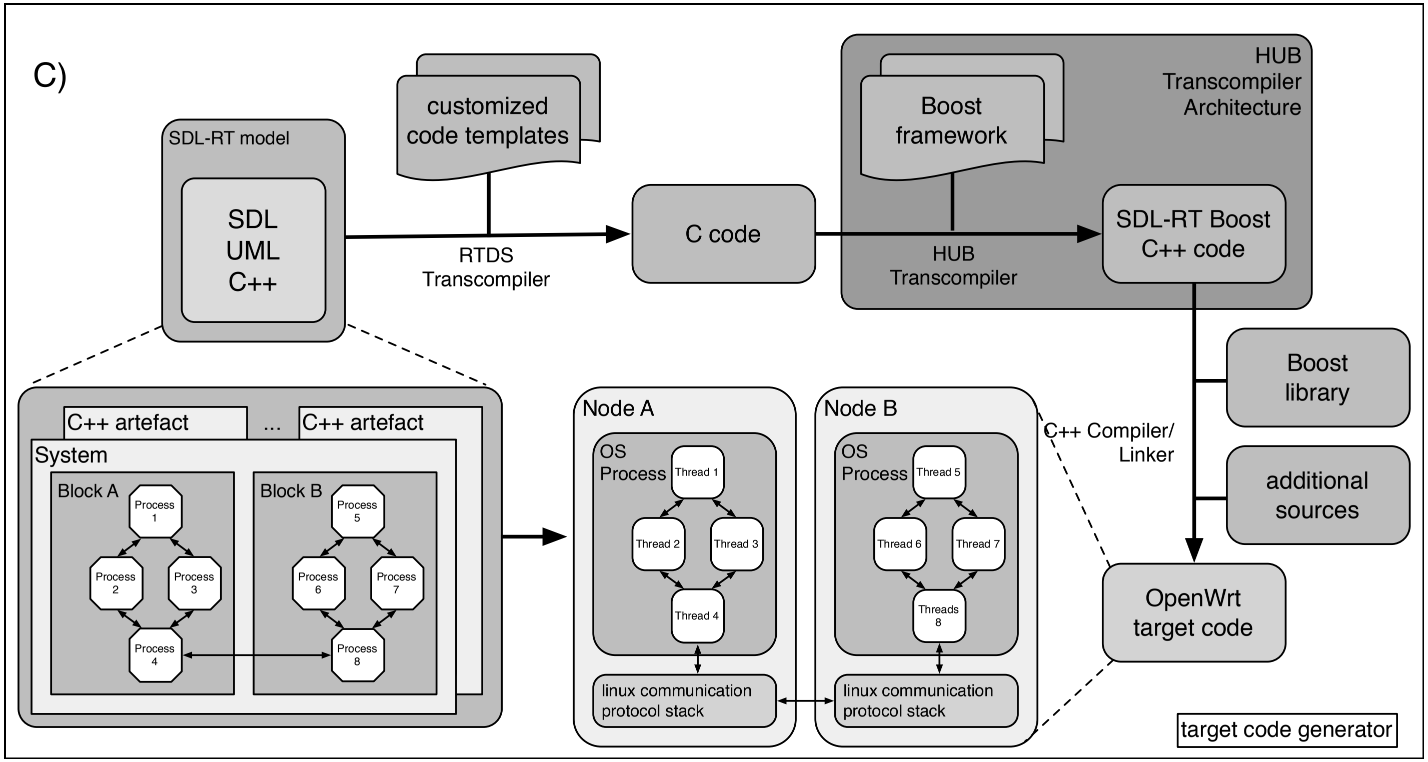


Fig. 2. C. Different code generation target types — OpenWrt-target generation

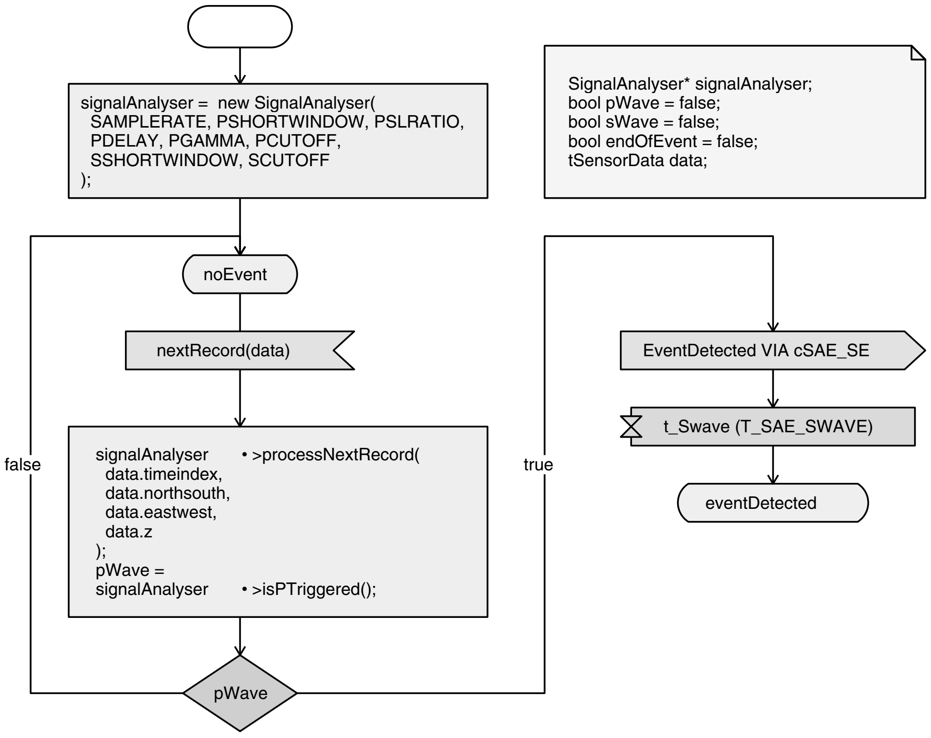


Fig. 3. Simplified SDL-RT state machine for Streaming Analysis

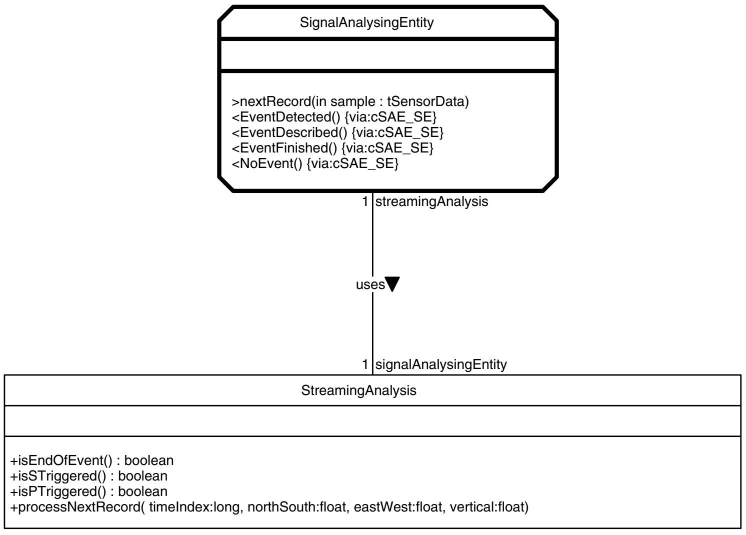

SignalAnalysingEntity
>nextRecord(in sample : tSensorData)
<EventDetected() {via:cSAE_SE}
<EventDescribed() {via:cSAE_SE}
<EventFinished() {via:cSAE_SE}
<NoEvent() {via:cSAE_SE}
1 streamingAnalysis
uses
1 signalAnalysingEntity
StreamingAnalysis
+isEndOfEvent() : boolean
+isSTriggered() : boolean
+isPTriggered() : boolean
+processNextRecord(timeIndex:long, northSouth:float, eastWest:float, vertical:float)

Fig. 4. SDL-UML specification for the Signal Analysing Entity process and the Streaming Analysis class

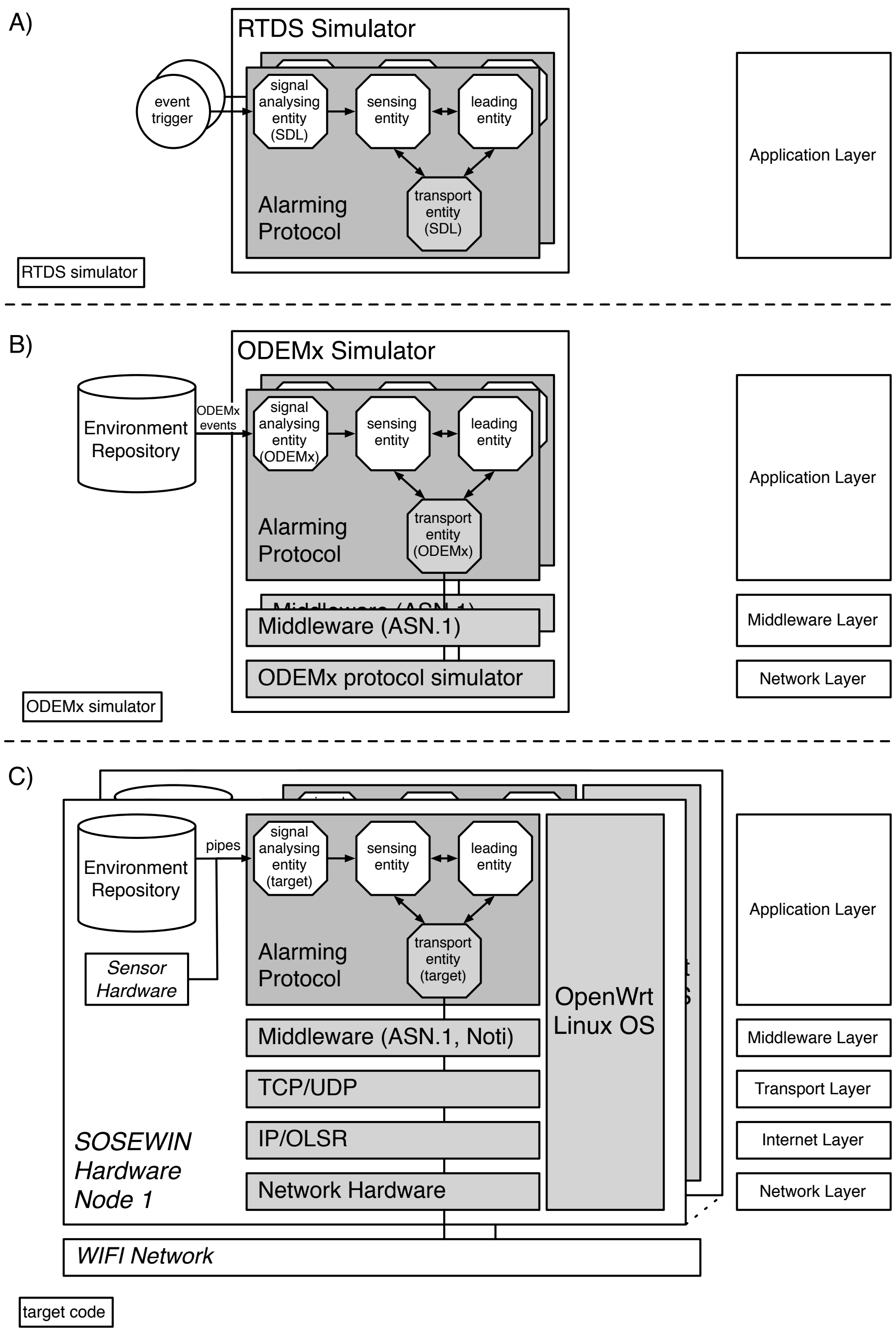


Fig. 5. The GAF4WSN simulators and testbeds

topology projects, a block needs to provide mechanisms for internal block communication. This implies a common base class. Unfortunately we were unable to distinguish SDL-RT blocks from user defined include files just by analyzing C code without loss of generality. Therefore we needed to introduce naming conventions for blocks and block classes if those should be instantiated dynamically. This doesn't break compatibility with the original SDL-RT, because compilation will still work even if the naming conventions are ignored.

ODEMx simulator/target code. Following the above principles, the modified compiler pipelines for an ODEMx simulator generation (Fig. 2.B) and for the target code generation (Fig. 2.C) can be constructed.

5.4 Experiment Management System

Experiment management ensures the consistency of artefacts that are necessary for an experiment, so that during analysis, the modeller is provided with all information that characterizes experiments: e.g. used model, parameter settings, results, characteristics of the simulator/prototype (code generation process, software version, algorithms used etc.) and the machine (account, operating system, execution time, etc.). Therefore EMS [25] provides support for the three phases: experiment planning, experiment execution and evaluation of the experiment results.

In our use case study the results of the simulator runs (event traces) or application-level-messages transmitted within the WSN are stored within the relational database, the Experiment Repository (by importing log files or direct access through an API). Experimental results can then be evaluated manually by the Visualizer. This tool allows the presentation of a P-wave travelling through the network, with its detection (or non-detection) being marked by the sensor nodes changing their colour.

But there are much more possibilities to evaluate experiment results such as visualizing the messages between software processes running on the Sensing Node as a UML sequence diagrams or doing statistical analysis, e.g. to compute the average message length of messages sent during a certain time period.

6 SOSEWIN Application

The main purpose of the SOSEWIN application we developed with our GAF4WSN is to perform the tasks of an EEWS (Sect. 3.1). Therefore in this chapter the basic principles of the Alarming Protocol (AP), implemented in the WSN application layer, are introduced. To go more into detail the Earthquake Synthesizer (ES) is explained, followed by a description of the integration of the signal analysis performed by a single node to detect a P-wave.

6.1 Cooperative Alarming Protocol

To establish a hierarchical alarming system, each node in the WSN runs the AP with different roles at runtime [26]. The SOSEWIN nodes are organized into clusters using criteria that determine the optimum communication efficiency. Each cluster is headed by a Sensing Node (SN) that is designated as a Leading Node (LN), with whom the other SNs within its cluster communicate general "housekeeping/status" information and initial alarms. The LN in turn communicates with other LNs, including issuing of system alarms, based on each LN knowing the status of the nodes that make up its respective cluster.

The SNs continuously monitor the ground shaking and perform a signal analysis with the goal to detect/trigger a P-wave. If a critical number of P-wave triggers have reached the cluster's LN, this node informs its neighbouring LNs. In the case that a LN has received enough cluster alarms, a so-called system alarm will be sent as fast as possible to the sinks in the SOSEWIN (Gateway Node (GN)) that are responsible for forwarding those alarms the potential end-users (e.g. public organizations responsible for disaster management).

6.2 Earthquake Synthesizer

The ES generates synthetic seismograms for each location of a node based on the method of Wang [27]. Synthetic seismograms offer the opportunity to test different methods of event detection and classification, with the freedom to introduce as much (or as little) "noise" to the data as required.

6.3 Integration of Streaming Analysis

The Signal Analysis (SA) [28] of the incoming sensor data stream is originally performed by algorithms developed at the GFZ using the Matlab program. These algorithms where re-implemented by hand into C++ to be highly efficient running on each SN. In the SDL model within the part of a SN this code is included as passive UML classes.

Figure 3 shows a simplified SDL-RT state machine representing the Streaming Analysis within the AP (the corresponding generated code for the target platform is presented in Listing 1). Figure 4 shows a combined specification of the SDL process und a passive UML class related to the `SignalAnalyser` class performing the signal analysing algorithms. It processes a data record provided by the SDL signal input queue via `nextRecord(data)`. If a P-wave is detected, it informs another state machine and goes into state `eventDetected`.

6.4 Simulators and Target Code

As motivated in Sect. 4 and described by their generation tools in Sect. 5.3 GAF4WSN supports code generation for SDL simulator, ODEMx simulator and target code. In Fig. 5 each subsection represents one of the three HUB SDL Transcompiler code generation target types and shows the specific input data, the generated parts of the AP, the substituted entities and the relation to the corresponding layers. In the development of the SOSEWIN software architecture each code generation target fulfils a specific purpose and provides a different level of abstraction.

```
#include <boost/thread/thread.hpp>
#include <boost/bind.hpp>
#include "SignalAnalysingEntity.h"

#define RTDS_PROCESS_NUMBER RTDS_process_SignalAnalysingEntity
#define RTDS_PROCESS_NAME SignalAnalysingEntity

SignalAnalysingEntity::SignalAnalysingEntity(RTDS::Logger& logger)
: SDLProcess(logger)
{
    RTDS_LOG_PROCESS_CREATION((int)msgQueue.writer, "SignalAnalysingEntity",
        (int)cover);
}

void SignalAnalysingEntity::main()
{
    RTDS_MSG_DATA_DECL
    short RTDS_transitionExecuted;

    SignalAnalyser* signalAnalyser;
    bool pWave = false;
    bool sWave = false;
    bool endOfEvent = false;
    tSensorData data;

    /* declare framework-side variables for the process
     * starts stdio.run() asynchronous stops when isRunnings
     * destructor is destroyed
     */
    boost::thread _workThread_t(boost::bind(
        &boost::asio::io_service::run, &ioService));
    auto_ptr<boost::asio::io_service::work> _workThread_ptr(isRunning);
    isRunning = 0;

    /* Initial transition */
    signalAnalyser =  new SignalAnalyser(
        SAMPLERATE, PSHORTWINDOW,
        PDELAY, PGAMMA,
        SSHORTWINDOW, SCUTOFF
    );
    RTDS_SDL_STATE_SET(noEvent);

    do
    {
        /* peek new message from queue */
        currentMessage = msgQRead();
        RTDS_LOG_MESSAGE_RECEIVE((int)&currentMessage->sender,
            (int)msgQueue.writer, currentMessage->sequenceNumber,
            getCurrentTime());
        /* Double switch state / signal */
        RTDS_transitionExecuted = 1;
        switch(RTDS_currentContext->sdlState)
        {
                /* ... */
                /* Transitions for state noEvent */
        case noEvent:
            switch(RTDS_currentContext->currentMessage->messageNumber)
            {
                /* Transition for state noEvent - message nextRecord */
            case nextRecord:
                RTDS_MSG_RECEIVE_nextRecord(data);
                signalAnalyser->processNextRecord(
                    data.timeindex,
                    data.northsouth,
                    data.eastwest,
                    data.z
                );
```

```
                    pWave = signalAnalyser->isPTriggered();
                    if ( pWave )
                    {
70                      RTDS_MSG_QUEUE_SEND_TO_NAME(EventDetected, 0, NULL,
                            "SensingEntity", RTDS_process_SensingEntity);
                        RTDS_SET_TIMER(t_Swave, T_SAE_SWAVE);
                        RTDS_SDL_STATE_SET(eventDetected);
                        break;
75                  }
                    RTDS_SDL_STATE_SET(noEvent);
                    break;
                default:
                    RTDS_transitionExecuted = 0;
80                  break;
                }
                break;
            default:
                RTDS_transitionExecuted = 0;
85              break;
            } /* End of switch(RTDS_currentContext->sdlState) */
            delete currentMessage;
        } while (1);
    }
```

Listing 1. Generated target C++ code for simplified Streaming Analysis state machine

The **SDL simulator** (Fig. 5.A) is used to test the functional behaviour of smaller ensembles of the SOSEWIN nodes. We abstract from concrete earthquakes, and underlying protocol layers. One further important preposition is perfect transmission behaviour of used communication channels over the air. The results of functional tests allow us to evaluate and improve the business logic of our AP. Typical outputs here are Sequence Diagrams (SD) and alarming signals without useful time constraints.

The **ODEMx simulator** (Fig. 5.B) uses the capability of our general-purpose ODEMx library, which supports next event simulation of parallel processes. It allows an integrated test of all developed software components (incl. SA, ASN.1 encoding and decoding) in a large network of sensor nodes. Every node processes its own set of sensor data provided by our ES. The outputs are general trace files which can be used to generate SD or ShakeMaps. Input, output and network configuration data come with precise time and spatial references. This simulator allows the estimation of required transmission times and transmission quality of alternative SOSEWIN configurations, which guarantees the early warning functionality is independent of different earthquake scenarios.

Target code (Fig. 5.C) is a generated software binary which is, after deployment and installation using the EMS, running on every node in the SOSEWIN network (Fig. 5.C shows two nodes communicating over WIFI network). For testing purposes it is possible to switch between the real sensor data input and the synthetic sensor data. The target code is integrated in a special package which could be installed remotely into the OpenWrt operating system.

Listing 1 shows the generated target C++ code based on the simplified Streaming Analysis state machine presented in Fig. 3. It realises the modeled state changes within an endless-running loop (lines 42-88) consuming signals depending on the current state. For example, if it has consumed a `nextRecord` signal (line 59) this is processed (line 61) and may result in the detection of a P-wave (line 68) and a switch to state `eventDetected` (line 73).

The **XEN simulator** [29] is an additional code generation target, not shown in Fig. 5, which allows to run several virtualized node hardware instances on a XEN server. XEN is a virtual machine monitor for different computer architectures. It allows several guest operating systems to be executed on the same computer hardware concurrently. In contrast to the SDL and ODEMx simulator the XEN simulator allows the complete integration test of all software components running on the real nodes (incl. the operating system and the complete communication stack). The link between the nodes is implemented by a network bridge without any packet loss. Due to the absence of a sensor in the virtualization the raw sensor data is provided by the Environment Repository.

7 Conclusion

SDL in combination with UML and ASN.1 is one of the most powerful technologies for modelling complex distributed reactive systems. In this paper we have demonstrated the extension of PragmaDev's RTDS into the core of model-driven development and administration approach of WSN, offered by one integrated framework GAF4WSN. Its architecture is based on OGC, OMG and ITU-T standards and combines different technologies for GIS, databases, behaviour modelling, information coding and code generation for target platforms and for different types of simulation. In addition, with experiment management, simulation and prototyping test cycles are handled independently, which leads to a quality improvement by applying these two evaluation techniques on different abstractions of the same real system. The advantages of GAF4WSN comprise not only the general support for modelling and simulation of environmental processes with geo-specific spatial contexts, but also the unification of information management of model experiments with tests of the prototyped WSN that allows us to use GAF4WSN for modelling as well as for administration. With the development of the first prototype of an EEWS for Istanbul we were able to demonstrate a proof of concepts. In the up-coming consolidation period of SOSEWIN in Istanbul we will extend our simulation experiments for preparing the extension of the real network with general network simulators like ns-3 to investigate large-scale topologies.

Acknowledgement

The work was supported by the SAFER project partners at the GFZ J. Zschau, C. Milkereit, K. Fleming and M. Picozzi, by our colleagues from the Humboldt University J.-P. Redlich, B. Lichtblau and J. Nachtigall (Systems Architecture Group) and our students T. Niebuhr, A. Jäkel, R. Kluth, D. Reinert and J. Wegener.

References

1. International Telecommunications Union: Recommendation Z.100 (11/07), Specification and Description Language (SDL), http://www.itu.int/rec/T-REC-Z.100/en
2. International Telecommunications Union: Recommendation X.690 (11/08), Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER), http://www.itu.int/rec/T-REC-X.690/en
3. Seismic eArly warning For EuRope (SAFER), http://www.saferproject.net
4. PragmaDev RTDS V3.4, http://www.pragmadev.com
5. The Network Simulator – ns-2, http://www.isi.edu/nsnam/ns
6. Henderson, T.R., Roy, S., Floyd, S., Riley, G.F.: ns-3 project goals. In: WNS2 '06: Proceeding from the 2006 workshop on ns-2: the IP network simulator. ACM Press, New York (2006)
7. JiST – Java in Simulation Time / SWANS – Scalable Wireless Ad hoc Network Simulator, http://jist.ece.cornell.edu
8. OPNET Modeler, http://www.opnet.com/products/modeler/home.html
9. Kuorilehto, M., Hännikäinen, M., Hämäläinen, T.D.: Rapid design and evaluation framework for wireless sensor networks. Ad Hoc Netw. 6(6), 909–935 (2008)
10. Park, S., Savvides, A., Srivastava, M.B.: Simulating networks of wireless sensors. In: WSC 2001: Proceedings of the 33nd conference on Winter simulation, pp. 1330–1338. IEEE Computer Society Press, Los Alamitos (2001)
11. Downard, I.: Simulating sensor networks in ns-2, Naval Research Laboratory (2003)
12. Levis, P., Lee, N., Welsh, M., Culler, D.: TOSSIM: accurate and scalable simulation of entire TinyOS applications. In: SenSys 2003: Proceedings of the 1st international conference on Embedded networked sensor systems, pp. 126–137. ACM, New York (2003)
13. Perrone, L., Nicol, D.: A scalable simulator for TinyOS applications. In: Proc. Winter Simulation Conference 2002, vol. 1, pp. 679–687. IEEE Computer Society Press, Los Alamitos (2002)
14. Kuhn, T., Geraldy, A., Gotzhein, R., Rothländer, F.: ns+SDL – The Network Simulator for SDL Systems. In: Prinz, A., Reed, R., Reed, J. (eds.) SDL 2005. LNCS, vol. 3530, pp. 103–116. Springer, Heidelberg (2005)
15. Milkereit, C., Fleming, K., Picozzi, M., Jäckel, K.H., Hönig, M.: Deliverable D4.5 Development of seismological analysis of single station recordings and network trigger for Early Warning purposes, http://casablanca.informatik.hu-berlin.de/wiki/images/4/41/D4.04_4.05_SAFER_GFZ.pdf
16. ShakeMap Manual; Technical Manual, Users Guide and Software Guide, http://pubs.usgs.gov/tm/2005/12A01/pdf/508TM12-A1.pdf
17. Erdik, M., Fahjan, Y., Ozel, O., Alcik, H., Mert, A., Gul, M.: Istanbul Earthquake Rapid Response and the Early Warning System. Bulletin of Earthquake Engineering 1(01), 157–163 (2003)
18. Picozzi, M., Milkereit, C., Zulfikar, C., Ditommaso, R., Erdik, M., Safak, E., Fleming, K., Ozel, O., Zschau, J., Apaydin, N.: Wireless technologies for the monitoring of strategic civil infrastructures: an ambient vibration test of the Faith Bridge, Istanbul, Turkey, http://casablanca.informatik.hu-berlin.de/wiki/images/2/23/Picozzi_et_al._Ambient_Vibration_Fatih_Bridge_BEE.pdf

19. Open Geospatial Consortium (OGC), `http://www.opengeospatial.org/standards/wfs`
20. SEEDLink, `http://www.iris.washington.edu/data/dmc-seedlink.htm`
21. Fischer, J., Kühnlenz, F., Eveslage, I., Ahrens, K., Nachtigall, J., Lichtblau, B., Heglmeier, S., Milkereit, C., Fleming, K., Picozzi, M.: Deliverable D4.22 Middleware for Geographical Applications, `http://casablanca.informatik.hu-berlin.de/wiki/images/2/2f/D4.22_SAFER_UBER.pdf`
22. ODEMx - Object oriented Discrete Event Modelling, `http://odemx.sourceforge.net`
23. Boost, `http://www.boost.org`
24. OpenWrt, `http://www.openwrt.org`
25. Kühnlenz, F., Theisselmann, F., Fischer, J.: Model-driven Engineering for Transparent Environmental Modeling and Simulation. In: Troch, I., Breitenecker, F. (eds.) Proceedings MathMod Vienna 2009 (2009) ISBN 978-3-901608-35-3
26. Fischer, J., Kühnlenz, F., Eveslage, I., Ahrens, K., Nachtigall, J., Lichtblau, B., Heglmeier, S., Milkereit, C., Fleming, K., Picozzi, M.: Deliverable D4.21 Develop Software for Network Connectivity, `http://casablanca.informatik.hu-berlin.de/wiki/images/7/73/D4.21_SAFER-UBER.pdf`
27. Wang, R.: A simple orthonormalization method for stable and efficient computation of Green's functions. Bulletin of the Seismological Society of America 89(3), 733–741 (1999)
28. Milkereit, C., Fleming, K., Picozzi, M., Jäckel, K.H., Hónig, M.: Deliverable D4.4 Development of the seismological analysis software to be implemented in the Low Cost Network, `http://casablanca.informatik.hu-berlin.de/wiki/images/4/41/D4.04_4.05_SAFER_GFZ.pdf`
29. XEN, `http://www.xen.org`

Modeling LTE Protocol for Mobile Terminals Using a Formal Description Technique

Anas Showk[1], David Szczesny[1], Shadi Traboulsi[1], Irv Badr[2], Elizabeth Gonzalez[1], and Attila Bilgic[1]

[1] Institute for Integrated Systems, University of Bochum, 44801 Bochum, Germany
{anas.showk,david.szczesny,shadi.traboulsi,elizabeth.gonzalez, attila.bilgic}@is.rub.de
[2] IBM Rational, Chicago, IL, USA
ibadr@us.ibm.com

Abstract. The Long Term Evolution (LTE) radio communication is the upgrade of the current 3G mobile technology with a more complex protocol in order to enable very high data rates. The usage of Model Driven Development (MDD) has arisen as a promising way of dealing with the increasing complexity of next generation mobile protocols. In this paper, a light version of the LTE protocol for the access stratum user plane is modeled using the SDL Suite^{TM} tool. The tool shows easy understanding of the model as well as easy testing of its functionality using simulation in cooperation with Message Sequence Chart (MSC). The simulation result shows that the implemented Specification and Description Language (SDL) guarantees a good consistency with the target scenarios. The system implementation is mapped to multiple threads and integrated with an operating system to enable execution in multi core hardware platforms.

Keywords: Service-Oriented applications, formal modeling, automatic code generation, formal verification, formal validation.

1 Introduction

Developing next generation mobile communication and wireless technologies greatly benefit from reusing prevailing approaches and best practices. For over a decade, most global installations have taken advantage of Model Driven Development (MDD) for communication products; oftentimes through tools using the Specification Description Language (SDL)[1]. This especially applies to most protocol based products developed within the last twenty years.

Besides other domains, a big majority of wireless base stations and personal handset devices world-wide currently use SDL with Testing and Test Control Notation version 3 (TTCN-3) [2]. Consequently, SDL generated, and TTCN tested systems now drive over eighty percent of all wireless technology in the world [3]. As the merits of domain specific modeling are just now becoming apparent, one has to commend the foresight used in development of the SDL language, over two decades ago. SDL, coupled with TTCN, has provided a solid workflow for model driven and Agile development approach for 2G and 3G wireless systems [4,5,6,7], and is slated to repeat the same for beyond 3G, and 4G wireless protocol development. Consequently, we use SDL to develop a light version of the Long Term Evolution (LTE) protocol stack layer 2 (L2) and layer 3 (L3) for the mobile terminal.

1.1 Development Methodology

The block diagram in Fig. 1 shows the implementation steps of the LTE protocol stack using SDL Suite^{TM}. In the top left side, the Message Sequence Chart (MSC) [8] editor is utilized to produce the target design for the LTE user plane which is compatible with the LTE standard [9]. The MSCs are used as guidance for graphical modeling in SDL and C functions implementation for header processing. After mapping the graphical model to C code, the generated code is linked and compiled with the hand written C implementation. The overall system is simulated to check the functionality and compare it with the design target MSCs. Further corrections of the LTE models can be done when needed.

As shown in the lower branch of Fig. 1, the LTE system implementation can be integrated into an operating system. The modeled system is divided into separate threads using the deployment editor. The multithread SDL model is converted to C code, compiled, and linked with the hand written code.

This paper is organized such that Sect. 2 gives an introduction to the LTE protocol stack and its data flow. Section 3 shows the LTE user plane target design. The development environment is presented in Sect. 4. After that, Sect. 5 discusses the SDL model of the system in detail. Section 6 is dedicated to the

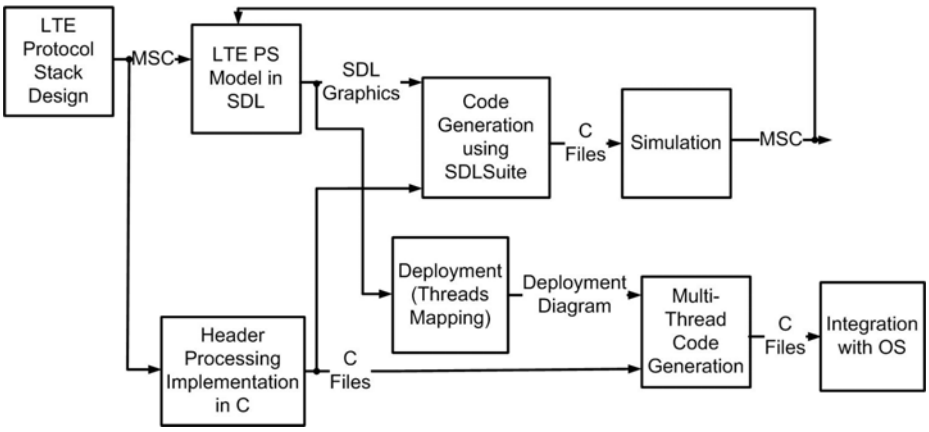

LTE Protocol Stack Design
MSC
LTE PS Model in SDL
SDL Graphics
Code Generation using SDLSuite
C Files
Simulation
MSC
Deployment (Threads Mapping)
Deployment Diagram
Multi-Thread Code Generation
C Files
Integration with OS
Header Processing Implementation in C
C Files

Fig. 1. Block diagram showing the LTE protocol stack implementation steps

integration with operating systems. In Sect. 7, the simulation results are presented followed by the conclusion and outlook to future research work in Sect. 8.

2 The LTE Protocol Stack

The LTE is the successor of the Universal Mobile Telecommunications System (UMTS) cellular technology. It enables much higher data rates to be achieved along with much lower packet latencies in an IP based system. The LTE will provide maximum data rates of 100 Mb/s in the downlink and 50 Mb/s in the uplink direction. Currently upcoming and future mobile devices enable the best quality of services in the current environment of the user with a future vision of an all IP based network. Another aspect is the dramatic increase of multimedia applications in the broadest sense including video streaming, video conferencing, and online gaming. Initiated in 2004, the LTE project focused on enhancing the Universal Terrestrial Radio Access (UTRA) and optimizing 3GPPs radio access architecture [9].

The LTE protocol stack L2 is divided into three sublayers, Medium Access Control (MAC), Radio Link Control (RLC), and Packet Data Convergence Protocol (PDCP). To summarize the flow of uplink data through all the protocol layers, an example with three IP packets is illustrated in Fig. 2. The PDCP performs IP header compression through Robust Header Compression (ROHC), followed by ciphering. A PDCP header is added, carrying information required

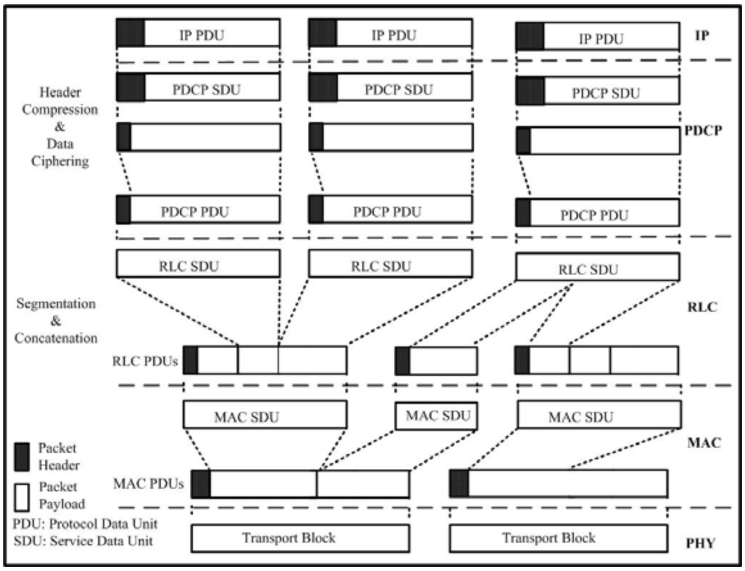


Fig. 2. The LTE protocol data flow in uplink direction from IP layer through L2 to the Physical layer (PHY)

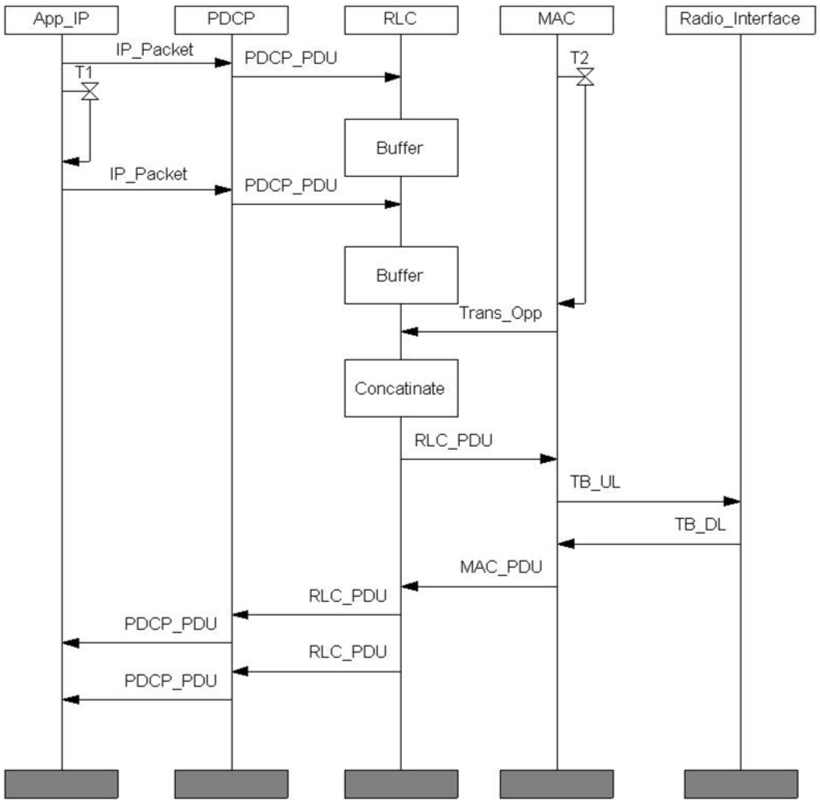


Fig. 3. MSC for the LTE data path in uplink and downlink showing packet flow from L3 through L2 and the transport block reception and processing in both layers

for deciphering in the mobile terminal. The output from the PDCP is fed to the RLC.

The RLC protocol performs concatenation and/or segmentation of the PDCP PDUs and adds an RLC header. The header is used for in sequence delivery (per logical channel) in the mobile terminal and for identification of RLC PDUs in case of retransmissions. Several RLC PDUs are forwarded to the MAC sublayer, which assembles them into a MAC Service Data Unit (SDU), and attaches the MAC header to form a transport block. The transport block size depends on the instantaneous data rate selected by the link adaptation mechanism. Thus, the link adaptation affects both the MAC and RLC processing. Finally, the physical layer attaches a Cyclic Redundancy Check (CRC) to the transport block for error detection purposes. The physical layer processing is performed before transmitting the signal to the air interface [10]. The inverse functionality is done in the downlink when receiving a transport block. More details about uplink and downlink processing are given in Sect. 5.

3 The LTE Protocol Stack Design

As a first step of our implementation, as shown in Fig. 1, the MSCs are designed. Several scenarios are plotted according to the LTE standard as described in the 3GPP Rel8 [9]. As an example, Fig. 3 demonstrates the MSC for a target LTE protocol scenario. The MSC represents a looped LTE protocol data path from the IP layer through L2 uplink to L2 downlink. Only the user plane is considered because its real time requirements are associated with longer execution time than the control plane. The MSCs are used for system analysis. The system can also be verified by comparing the designed MSCs with the MSCs generated by SDL Suite^{TM} simulator User Interface (UI).

The application data rate is realized using the timer *T1* and the IP packet payload length. In the uplink direction (see the upper part of Fig. 3) the IP packet should be processed in the PDCP sublayer and sent to the RLC sublayer to be concatenated with other PDCP PDUs depending on the transmission opportunity notification from MAC sublayer. The data rate in the mobile terminal is determined by the timer T2 together with the transport block size.

In the downlink, when MAC sublayer receives a transport block it should process its header and forward the RLC PDUs to the upper sublayers (see Fig. 3). The PDCP PDUs should be extracted from the received RLC PDUs and sent to PDCP sublayer to feed the IP layer with the IP packets.

4 SDL Environment

Formal Description Techniques (FDTs) are efficient in specifying complex communication protocols. FDTs guarantee syntactically and semantically unambiguous formal descriptions of communication protocols. In addition, they also guarantee interoperable and compatible implementations of these protocols independent of their implementers. This removes a lot of the anxiety of current software vendors to make their stack interoperable with other stacks developed by other vendors. Furthermore, the conformance of these protocols to a given standard can be checked with the help of predefined tests [11].

SDL supports object oriented software design by dedicated elements of the language in contrast to other FDTs. Furthermore, its quick to learn graphical notation is self documenting and thus easily maintainable [12]. These are the reasons why SDL was chosen to model the LTE protocol stack.

SDL is the most widely used FDT in the area of telecommunications. The basic theoretical model of an SDL system consists of a set of Extended Finite State Machines (EFSM) that run in parallel. These machines are independent of each other and communicate with asynchronous discrete signals. The LTE protocol stack model consists of the components described in the following sub sections.

4.1 SDL Structure

The LTE Protocol Stack (PS) is structured using the hierarchical decomposition with system, block, process, and procedure as the main building components. The main hierarchical levels in SDL are utilized to organize the LTE modeling. Every PS sublayer is represented by a sub block (that is MAC, RLC, and PDCP). The sub block in turn, is divided into sub sub blocks or processes according to the functionality or entities of the target layer. Some functionality like header building/processing is implemented using external C functions to have more efficient code for accessing memory and doing other hardware closer functionalities.

4.2 SDL Communication

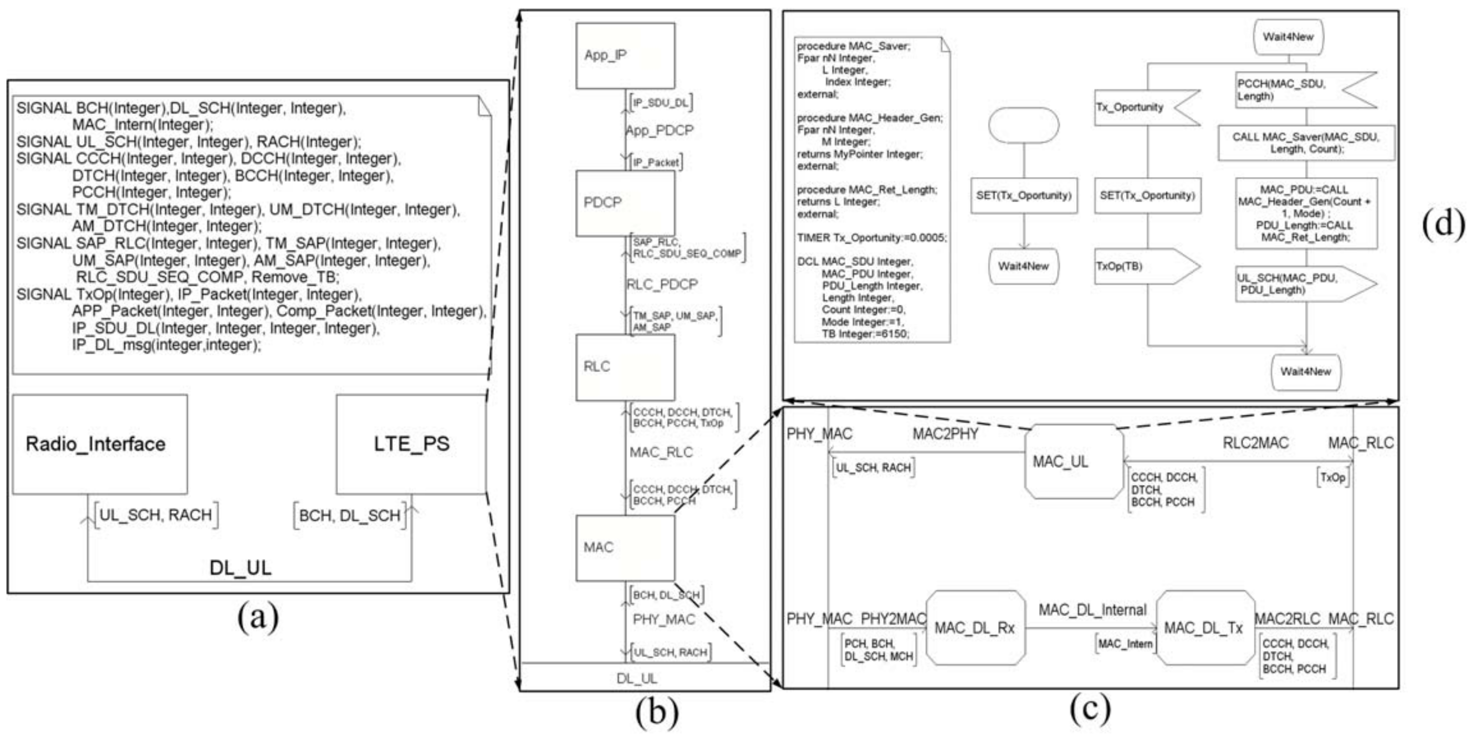
In SDL, global data is accessed by an exchange of signals: the language model does not allow direct reading from or direct writing to data owned by an enclosing block. Communication requires that information between processes, or between processes and the environment, is sent with signals (optionally with parameters). SDL signals are sent asynchronously: the sending process continues executing without waiting for an acknowledgment from the receiving process. PDUs are sent as signals. Synchronous communication is possible via a shorthand: remote procedure call. This shorthand is transformed to signal sending and waiting for a signal for the acknowledgment [1].

4.3 SDL Behavior

The dynamic behavior in an SDL system is described in the processes using EFSM. The system/block hierarchy is a static description of the system structure. Processes in SDL can be created at system start, or created and terminated dynamically at runtime. More than one instance of a process can exist. Each instance has a unique process identifier (Pid). This makes it possible to send signals to individual instances of a process. The concept of processes and process instances that work autonomously and concurrently makes SDL a true real time language [13]. The other advantage of the processes concurrency that it makes the parallelism easier to identify and exploit than the pure C programming.

5 The LTE Protocol Stack Model

The LTE protocol stack model in the SDL system is composed of two blocks called *LTE_PS* and *Radio_Interface*, as illustrated in Fig. 4(a). The radio interface forwarding functionality is modeled using two processes. The first for receiving the transport block and the other for forwarding the transport block to



```
SIGNAL BCH(Integer),DL_SCH(Integer, Integer),
        MAC_Intern(Integer);
SIGNAL UL_SCH(Integer, Integer), RACH(Integer);
SIGNAL CCCH(Integer, Integer), DCCH(Integer, Integer),
        DTCH(Integer, Integer), BCCH(Integer, Integer),
        PCCH(Integer, Integer);
SIGNAL TM_DTCH(Integer, Integer), UM_DTCH(Integer, Integer),
        AM_DTCH(Integer, Integer);
SIGNAL SAP_RLC(Integer, Integer), TM_SAP(Integer, Integer),
        UM_SAP(Integer, Integer), AM_SAP(Integer, Integer),
         RLC_SDU_SEQ_COMP, Remove_TB;
SIGNAL TxOp(Integer), IP_Packet(Integer, Integer),
        APP_Packet(Integer, Integer), Comp_Packet(Integer, Integer),
        IP_SDU_DL(Integer, Integer, Integer, Integer),
        IP_DL_msg(integer,integer);
```

```
procedure MAC_Saver;
Fpar nN Integer,
     L Integer,
      Index Integer;
external;

procedure MAC_Header_Gen;
Fpar nN Integer,
     M Integer;
returns MyPointer Integer;
external;

procedure MAC_Ret_Length;
returns L Integer;
external;

TIMER Tx_Oportunity:=0.0005;

DCL MAC_SDU Integer,
    MAC_PDU Integer,
    PDU_Length Integer,
    Length Integer,
    Count Integer:=0,
    Mode Integer:=1,
    TB Integer:=6150;
```

Fig. 4. The LTE protocol model hierarchy levels in SDL: (a) SDL System, (b) The LTE protocol block, (c) MAC sub block, and (d) MAC uplink process

the downlink. As illustrated in Fig. 4(b), inside the *LTE_PS* block, there are four sub blocks, and 20 processes used to model the LTE protocol stack L2 and L3. All the models are implemented according to the LTE standard [14,15,16]. More details of the process modeling is given in the next subsections.

5.1 IP Packet Generation

The IP packet generation consists of two processes: one to emulate an application transmitting data and the other to build the IP header. The IPv4 [17] header with a length of 20 bytes is used in this implementation. The functionality is implemented using an external C function, which returns the pointer to the IP packet, and is invoked inside the SDL process. The IP packet length and the pointer to the IP packet are forwarded to L2 by an SDL signal for further processing and header building.

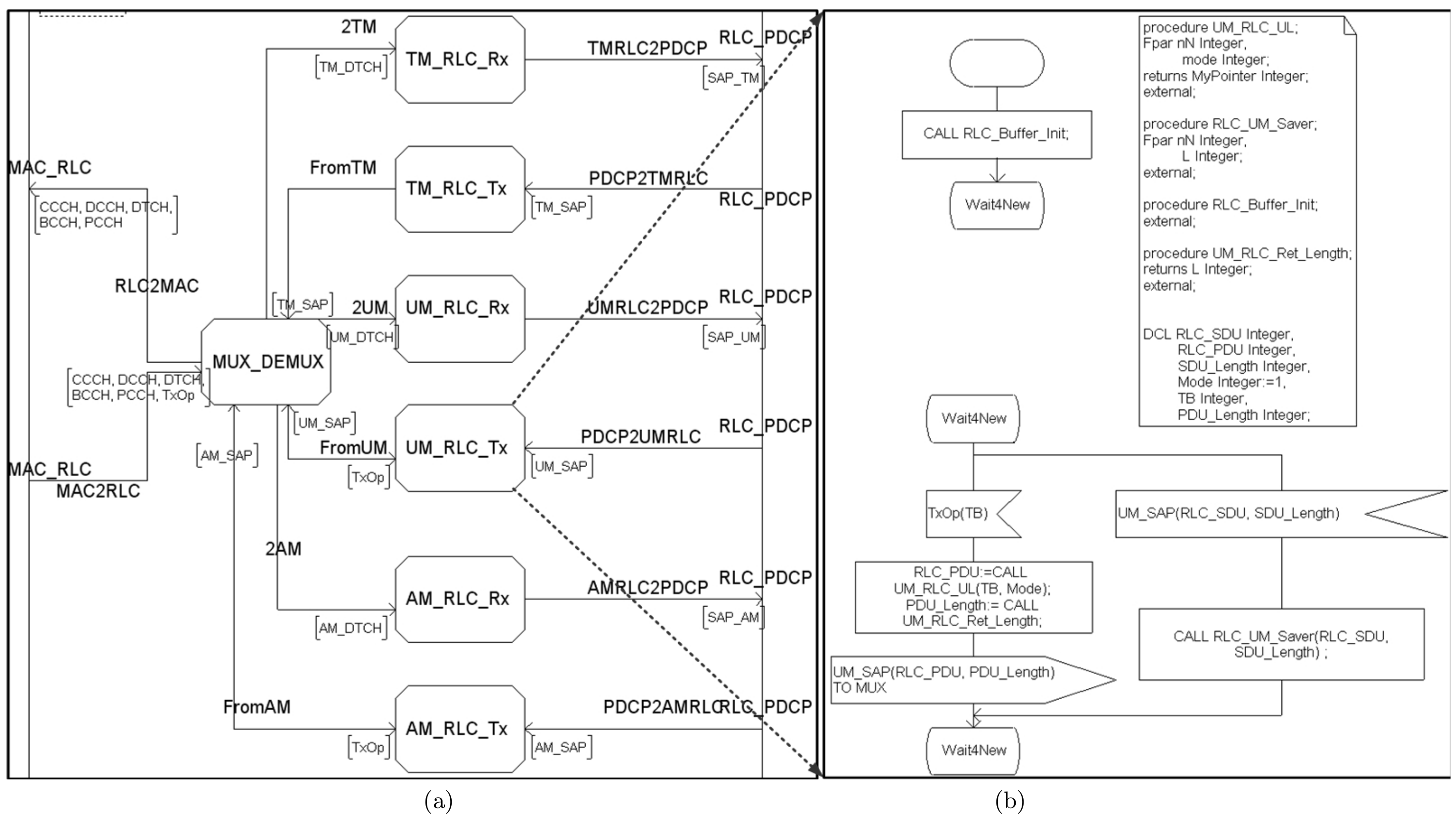
5.2 PDCP Uplink

In the *PDCP* sub block, there are two processes used to model the LTE PDCP uplink. The first process receives the pointer to the IP packet from the upper layer. The Robust Header Compression (ROHC) and data ciphering are not included in this implementation because they are already implemented and tested with 3G system. The components with names relevant to the above tasks are not fully implemented but used only to forward the received packet. For every IP packet received from the upper layer, the second process appends a PDCP header and increments the sequence number counter. Then the pointer to the PDCP PDU is forwarded to the RLC sublayer.

5.3 RLC Uplink

The RLC sublayer has three different modes: Transparent Mode (TM), Unacknowledged Mode (UM) and Acknowledged Mode (AM). In the TM the RLC entity does not concatenate RLC SDUs nor add RLC headers, but forwards the received RLC SDUs to the MAC sublayer. On the other hand, when the RLC operates in the UM or AM, the RLC entity concatenates and/or segments the received RLC SDUs and adds the RLC headers to build the RLC PDUs. Then, the RLC PDUs are forwarded to the MAC sublayer. The main difference between AM and UM is that in the AM, the acknowledgement should be received from the receiving entity. The resegmentation can be done in the AM mode as well.

There are seven processes used to model the RLC functionalities in all three modes as shown in Fig. 5(a). The RLC entity saves the received RLC SDUs (that is, the PDCP PDUs) in a buffer and waits for the transmission opportunity indicated by the MAC sublayer. When receiving an indication from the MAC entity to transmit, combined with the length of the transport block allowed to be sent, the RLC SDUs are concatenated according to the transport block


```
procedure UM_RLC_UL;
Fpar nN Integer,
        mode Integer;
returns MyPointer Integer;
external;

procedure RLC_UM_Saver;
Fpar nN Integer,
        L Integer;
external;

procedure RLC_Buffer_Init;
external;

procedure UM_RLC_Ret_Length;
returns L Integer;
external;

DCL RLC_SDU Integer,
        RLC_PDU Integer,
        SDU_Length Integer,
        Mode Integer:=1,
        TB Integer,
        PDU_Length Integer;
```

Fig. 5. SDL modeling of the RLC sublayer: (a) RLC sub block showing all processes for uplink and downlink and (b) Unacknowledged mode uplink process

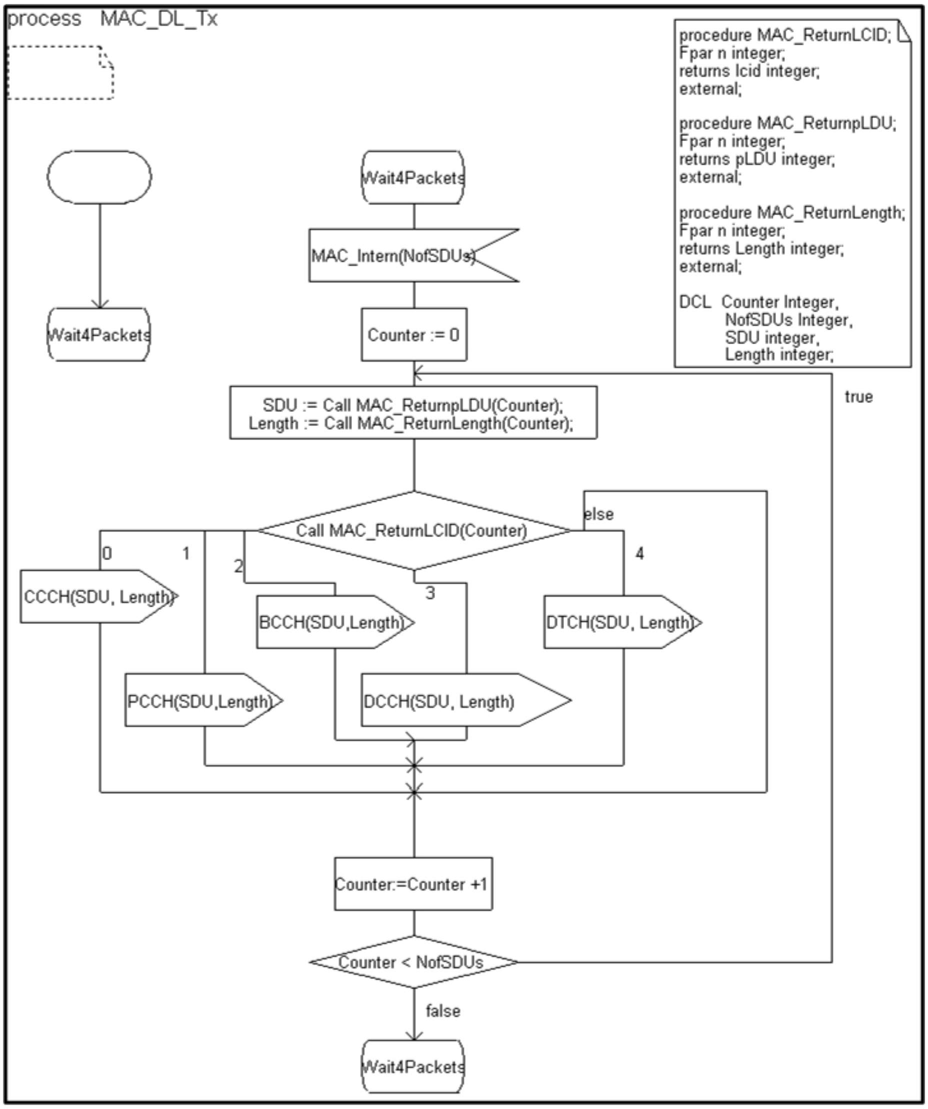


Fig. 6. SDL modeling of MAC DL shows transmitting MAC SDUs to different logical channels according to the LCID field in the received header

length to build the RLC PDU. In the TM entity (the *TM_RLC_Tx* process) no RLC header is built. On the other hand, the UM and AM transmitter processes build the RLC header and append it to the concatenated SDUs using external C functions. A pointer to the RLC PDU and its length are forwarded to the MAC sublayer, via the *MUX_DEMUX* process using the Dedicated Traffic Channel (DTCH), as shown in Fig. 5(b).

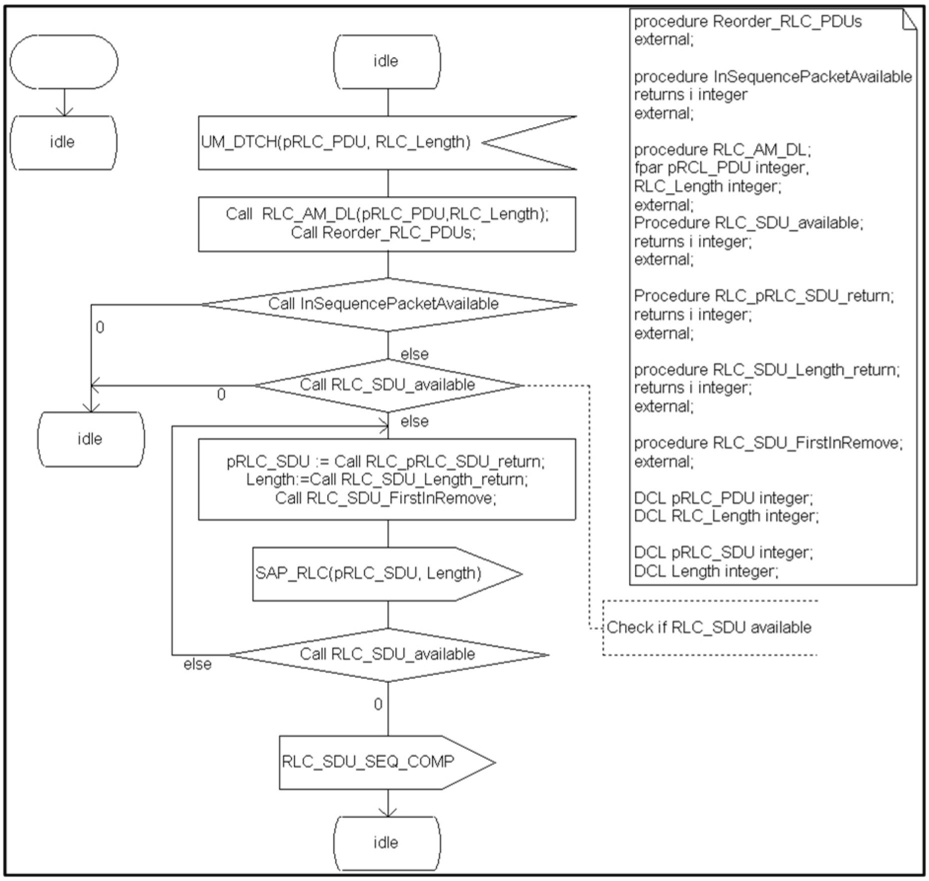

idle
idle
UM_DTCH(pRLC_PDU, RLC_Length)
Call RLC_AM_DL(pRLC_PDU,RLC_Length);
Call Reorder_RLC_PDUs;
Call InSequencePacketAvailable
0
else
Call RLC_SDU_available
0
idle
else
pRLC_SDU := Call RLC_pRLC_SDU_return;
Length:=Call RLC_SDU_Length_return;
Call RLC_SDU_FirstInRemove;
SAP_RLC(pRLC_SDU, Length)
Call RLC_SDU_available
else
0
RLC_SDU_SEQ_COMP
idle
procedure Reorder_RLC_PDUs
external;
procedure InSequencePacketAvailable
returns i integer
external;
procedure RLC_AM_DL;
fpar pRCL_PDU integer,
RLC_Length integer;
external;
Procedure RLC_SDU_available;
returns i integer;
external;
Procedure RLC_pRLC_SDU_return;
returns i integer;
external;
procedure RLC_SDU_Length_return;
returns i integer;
external;
procedure RLC_SDU_FirstInRemove;
external;
DCL pRLC_PDU integer;
DCL RLC_Length integer;
DCL pRLC_SDU integer;
DCL Length integer;
Check if RLC_SDU available

Fig. 7. SDL modeling of RLC DL shows reordering RLC SDUs and send them to the upper layers

5.4 MAC Downlink

The received MAC PDU header is decoded using a C function. As a consequence, the number of multiplexed MAC SDUs, the logical channel identifier (LCID), and the length of every MAC SDU are used to forward the received SDUs to the higher layer via the dedicated logical channel as shown in Fig. 6. There are two processes used to model the above functionalities (see Fig. 4(c)). The first process decodes the received packet header and sends the number of MAC SDUs to the second process which in turn transmits every MAC SDU to the RLC sublayer via logical channels. The traffic data is sent via the DTCH.

5.5 RLC Downlink

The pointer to the RLC PDU is received by the *MUX_DEMUX* process and forwarded to the corresponding process, depending on the RLC entity mode. In the TM, the RLC PDU is forwarded to the PDCP layer without any processing according to [14]. On the other hand, in UM and AM, the RLC entity decodes the received header to check the starting address, the sequence number, and the length of every RLC SDU. As illustrated in Fig. 7, in the case that the SDUs are received out of order, the reordering is carried out and sent to the PDCP sublayer. If there is more than one copy of the same packet received correctly, only one will be processed and the other(s) will be discarded. The main difference between AM and UM is the retransmission of the packet which is not received at all or corrupted in the AM. As a consequence, the error correction using automatic repeat request (ARQ) is done in the acknowledged mode (see [14]). The ARQ procedure is not implemented in this version and a unidirectional channel is assumed.

5.6 PDCP Downlink and IP Packet Reception

In the downlink direction, the received RLC SDUs represents PDCP PDUs. The header of the PDCP PDU is decoded and the sequence number and packet length are used for in sequence delivery of the packets to L3. The duplicated packets are eliminated as well. After removing the PDCP header, the pointer to the beginning of the PDCP SDU is sent to the IP layer.

In the IP layer only the IP header checksum is calculated and evaluated for the received IP packet. The IPv4 header of length 20 bytes is removed such that the payload data can be sent to the application.

6 Integration with Operating Systems

The LTE system implementation is integrated into an operating system. The modeled system is divided into separate threads using the deployment editor. In our setup, every SDL process acts as one thread using multiplicity (*) on the aggregation between a component and the thread that contains the SDL block as illustrated in Fig. 8. The multi threaded model is converted to C code, compiled and linked with the hand written C code. With the threaded integration, SDL Suite^{TM} supports different operating systems like POSIX, Win32, VxWorks, and Nucleus Plus. As a consequence, the generated C code can be integrated with one of those operating systems. The integration with POSIX is done by taking the advantage of the standard *pthread* library for initiating, creating, starting and managing threads. The concurrency of processes makes the parallelism is easier to identify and exploit than the imperative programming style such as C.

We run the generated code of the designed LTE stack model on an embedded ARM11 processor with 32 KB cache size running at a frequency of 350 MHz. The observed average execution time for uplink and downlink processing is around 70 microseconds each. The development efforts for the modeling described in Sect. 5 and the integration with operating system is about 6 man/months including the learning phase for LTE and SDL.

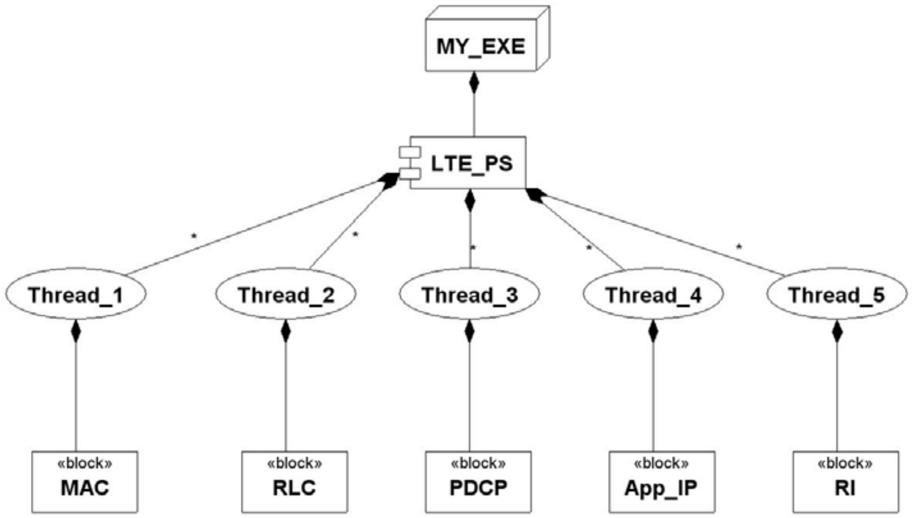


Fig. 8. The deployment diagram for the LTE protocol stack shows the multi-threading of the modeled system

7 Simulation Results

IBM® Rational® SDL Suite^{TM} is a real time, software development solution that provides specification and development capabilities for complex, event driven communications systems and protocol software. The SDL Suite^{TM} v6.1 is used to analyze the LTE model and to automatically generate C code. The generated code is compiled and linked with the hand written C implementation for header processing (see Fig. 1). The overall system is simulated to check the functionality and compare it with design target MSCs presented in Sect. 3.

As illustrated in Fig. 9, the IP packets propagate through PDCP, RLC and MAC sublayers to the radio interface. The SDL processes which are not relevant to the uplink are removed from the MSC for the sake of clarity.

The received transport block is processed in MAC, RLC, PDCP, and IP layers to extract the IP packet payload from it (see Fig. 10). The MSC in Fig. 9 and Fig. 10 are comparable to the target design MSC shown in Fig. 3. The main difference is the increased number of entities in the simulation output, because in SDL every process is treated as a separate entity. The SDL processes which are not relevant to the downlink are removed from the MSC for the sake of clarity. From a functionality point of view, the LTE protocol considered in this work is successfully implemented using SDL.

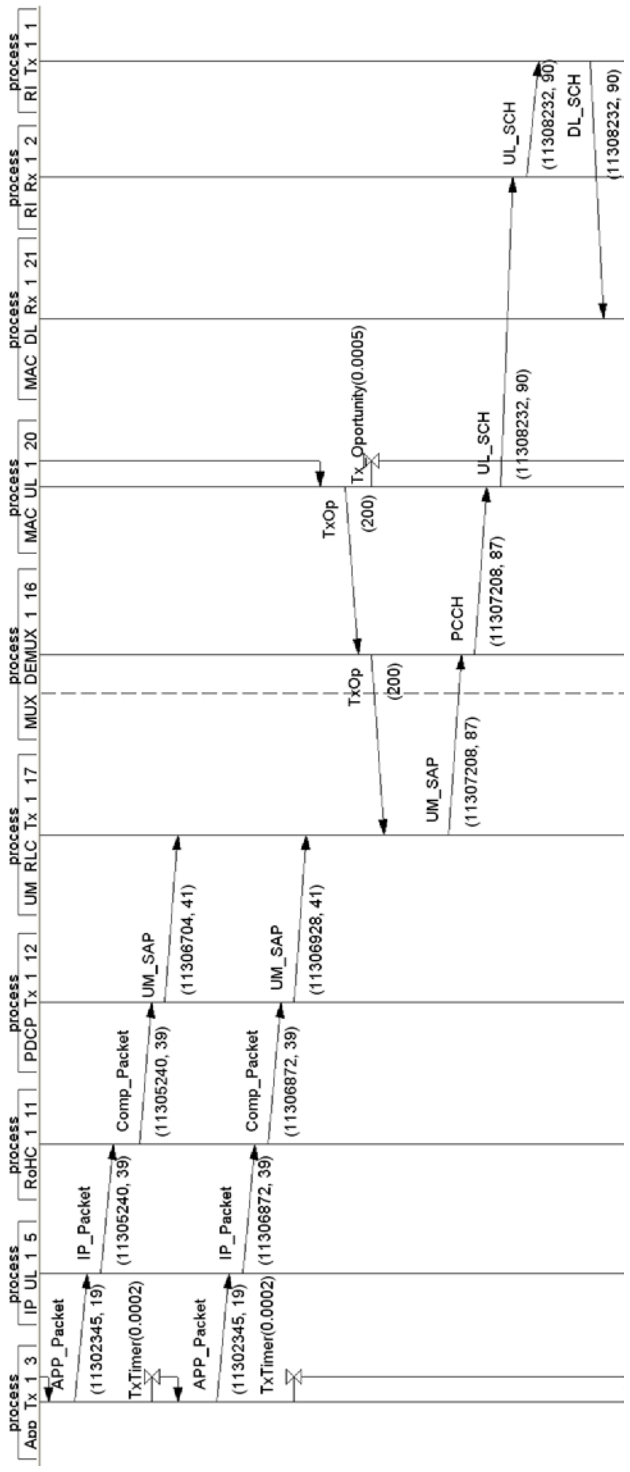


Fig. 9. The MSC trace as a simulation output shows the packet propagation through L2 and L3 for uplink

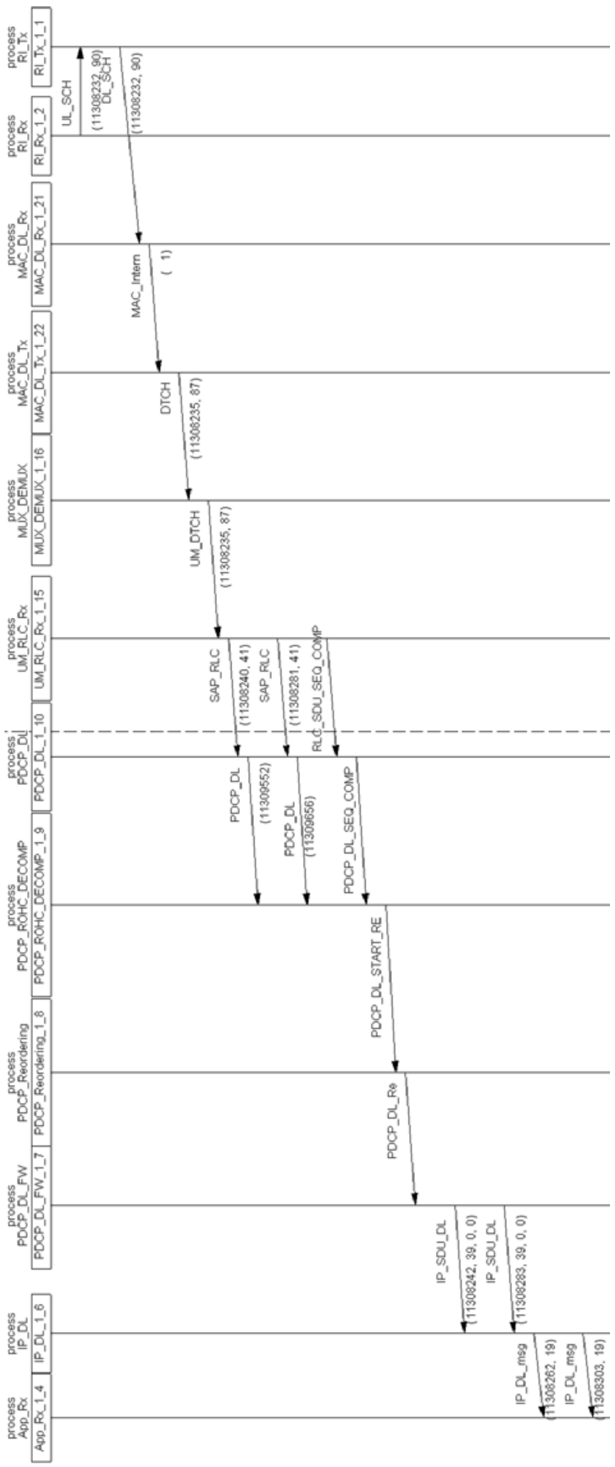


Fig. 10. The MSC trace as a simulation output shows the packet propagation through L2 and L3 for downlink

8 Conclusion

In this paper, we present efficient modeling of a light version of the LTE protocol stack in the UE side using SDL together with hand written C code. The overall system is simulated to check the functionality and compare it with design target MSCs which is compliant with 3GPP release 8 standard of the LTE. The simulation result shows that the implemented SDL guarantees a good consistency with the target test scenario, and has not any mismatching of logic flows as well as semantic errors.

The SDL model is decomposed into threads to enable execution in multi core as well as in single core platforms. For future work, we plan to extend the introduced LTE implementation, integrate it with any POSIX like operating system that support multi core and run them on a multi core embedded system.

Acknowledgments. The authors acknowledge the excellent cooperation with all partners within the ICT eMuCo project and the support by the European Commission. Further information is available on the project web site: `http://www.emuco.eu`.

References

1. International Telecommunications Union: Recommendation Z.100 (11/07), Specification and Description Language (SDL), `http://www.itu.int/rec/T-REC-Z.100/en`
2. International Telecommunications Union: Recommendations Z.161 to Z.170 all (11/2007) except Z.167 and Z.168 (11/2008), Testing and Test Control Notation version 3: TTCN-3, `http://www.itu.int/rec/T-REC-Z.161/en`, `http://www.itu.int/rec/T-REC-Z.170/en`
3. Badr, I.: Proven Approach for Developing Next Generation Communications Systems. ESE Magazine 15(6), `http://www.esemagazine.com/pdf/dl_pdf/ESE-sep07.pdf`
4. Hännikäinen, M., Knuutila, J., Hämäläinen, T., Saarinen, J.: Using SDL for Implementing a Wireless Medium Access Control Protocol. In: Proceedings of the 2000 International Conference on Microelectronic Systems Education, pp. 229–236. IEEE Computer Society, Washington DC (2000)
5. Park, S.-G., Shin, Y.-S., Kim, D.Y.: Design and Implementation of Protocols Related to Radio Network Layer Over eNB IN 3GPP LTE System. In: 66th IEEE Vehicular Technology Conference, pp. 194–199. IEEE Computer Society, Washington DC (2007)
6. Álvarez, J.M., Cámara, P.-d.-l., Martínez, J., Merino, P., Pérez, F.C., Morillo, V.: An SDL Implementation of the UMTS Radio Resource Control Protocol Oriented to Conformance Testing. In: Proceedings of the Second International Symposium on Leveraging Applications of Formal Methods, Verification and Validation, pp. 397–401. IEEE Computer Society, Washington DC (2006)
7. Jung, K.-R., Choi, J., Song, P., Nam, Y.-H.: Design and Implementation of a Radio Resource Control Protocol in WCDMA using SDL. In: IEEE Vehicular Technology Conference (VTC) 54th, vol. 2, pp. 990–994. IEEE Computer Society, Washington DC (2001)

8. International Telecommunication Union: Recommendation Z.120 (04/04), Message sequence chart (MSC), http://www.itu.int/rec/T-REC-Z.120/en
9. 3rd Generation Partnership Project (3GPP): The LTE Protocol Specification, 3GPP Rel8, http://www.3gpp.org/Release-8
10. Dahlman, E., et al.: 3G Evolution: HSPA and LTE for Mobile Broadband, 1st edn. Academic Press, London (2007)
11. Steppler, M.: Performance Analysis of Communication Systems Formally Specified in SDL. In: Proceedings of the 1st International Workshop on Software and Performance (WOSP), pp. 49–62 (1998) ISBN:1-58113-060-0
12. Hogrefe, D.: Estelle, LOTOS und SDL-Standard Spezifikationssprachen für verteilte Systeme. Springer, Heidelberg (1989)
13. IBM® Rational®, S.D.L.: Suite^{TM} User Manual, SDL Suite^{TM} v6.1
14. 3GPP TS 36.321: Evolved Universal Terrestrial Radio Access (E UTRA); Medium Access Control (MAC) Specification, http://www.3gpp.org/ftp/Specs/html-info/36-series.htm
15. 3GPP TS 36.322: Evolved Universal Terrestrial Radio Access (E UTRA); Radio Link Control (RLC) protocol specification. For URL see [14]
16. 3GPP TS 36.323: Evolved Universal Terrestrial Radio Access (E UTRA); Packet Data Convergence Protocol (PDCP) Specification. For URL see [14]
17. Internet Society, The Internet Engineering Task Force: RFC 791: Internet Protocol DARPA Internet Program Protocol Specification (September 1981), http://www.ietf.org/rfc/rfc0791.txt

SDL/Virtual Prototype Co-design for Rapid Architectural Exploration of a Mobile Phone Platform

Shadi Traboulsi, Felix Bruns, Anas Showk, David Szczesny, Sebastian Hessel, Elizabeth Gonzalez, and Attila Bilgic

Institute for Integrated Systems, Ruhr-University of Bochum
D-44780 Bochum, Germany
{shadi.traboulsi,felix.bruns,anas.showk,david.szczesny,sebastian.hessel, elizabeth.gonzalez,attila.bilgic}@is.rub.de

Abstract. In this paper we present a new hardware/software co-design methodology for embedded systems, where software components written in Specification and Description Language (SDL) execute on a soft-model of a hardware platform, a so called Virtual Prototype (VP). The proposed approach enables fast exploration of different hardware and software design options at high level of abstraction in order to make early system design decisions. We prove our approach by considering the Long Term Evolution (LTE) communication stack as a use case for the architectural exploration of our mobile terminal. The open source L4/Fiasco microkernel is deployed as a Real-Time OS to run the modem application represented by the LTE SDL-modelled protocol stack. We profile and analyze the system performance by measuring average and maximum packet processing times under various hardware and software conditions. Thereby, we are able to rapidly obtain an efficient design point that provides 80 % packet processing speedup against other unoptimized implementations while meeting the required timing constraints and maintaining a good balance between area and power consumption.

Keywords: hardware/software co-design, rapid system prototyping, design-space exploration, mobile terminal, SDL.

1 Introduction

The complexity of embedded systems is increasing over time due to the integration of more and more functionalities into a single chip, which is supported by the advances in fabrication technology. Nowadays, these systems are made up of heterogeneous architectures consisting of a broad range of IP modules like embedded processors, accelerator blocks, interface modules, a memory subsystem, and a communication infrastructure through which these blocks can interact for the exchange of data and synchronization.

Embedded systems like mobile phones have serious power constraints because of their limited battery life time. Moreover, these systems often have real-time characteristics. For instance, communication devices have to process a certain number of packets per second in order to guarantee no degradation in the system's efficiency and quality of service. In addition, reducing the chip area is an important aspect in chip design due to its impact on chip cost. The metrics introduced above (power, performance and area) influence the selection of an appropriate system architecture based on given constraints and their trade-off figures. System constraints, together with the degree of system's complexity, determine the effort needed to optimize hardware and software components of the system.

To be able to define system bottlenecks at an early stage of system development, describing hardware and software at high level of abstraction is mandatory. SystemC, a recent hardware modelling language, has gained a lot of attention for describing hardware components at high level of granularity for the purpose of design exploration [1]. On the software side, formal description techniques such as the Specification and Description Language (SDL) provide simplicity and modularity needed to cope with complex modern software applications.

In this paper we present a new methodology that bridges software and hardware development as both impact the fulfilment of system constraints. The proposed methodology combines modular and abstract software design using SDL with system-level hardware design based on virtual prototyping (VP). This allows for optimizing and customizing the whole system at a high level of abstraction, where simulations are faster and the potential for optimization is larger compared to low level detailed implementations.

The rest of the paper is organized as follows. In the following section we give an overview about co-design and modelling methods from literature. Section 3 describes our SDL/VP co-design approach. An abstract design of a mobile phone system with its VP and software stack is demonstrated in Sects. 4 and 5. As a case study, we demonstrate in Sect. 6 the usage of the proposed methodology for architectural exploration of the reference mobile system. Section 7 concludes the paper and gives an outlook on future work.

2 Related Work

There is currently much research being done in the area of Hardware/Software (Hw/Sw) co-design. Many approaches have been proposed that rely on different concepts. Interplay of processes describing a whole system is modelled in [2] using performance networks, where system workload and services of resources are described as event and resource streams, respectively. These two stream types interact in performance components that are connected to a network to analyze resource loads. This method can only be used for initial steps of system partitioning and mapping of tasks into specific resources. It is very abstract and does not provide any mechanism for evaluating the impact of hardware and software variations on system performance.

Several co-design tools are developed throughout research. Cosyma [3] is an environment that enables a C-based description of an embedded system's tasks and their interconnection. It transforms the system behaviour into an Extended Syntax (ES) graph, which is then used for determining the mapping of system tasks into software and hardware resources. Software parts are converted to C code and hardware blocks are generated in a HardwareC language. Estimations for both software and hardware metrics are then obtained by simulating the object code with Register Transfer Level (RTL) hardware models and their synthesized counterparts, respectively. A similar approach is followed in [4], where hardware partitions are memory-mapped with an interface through interrupt lines to the controlling software and estimations are computed in cycles per byte for comparison with predefined cost and performance parameters. Another work in [5] uses Co-design Finite State Machines (CFSM) to describe both hardware and software parts of a system. Iterative evaluation is then performed for the sake of system partitioning. The next step involves conversion of software partitions into C routines and mapping the hardware parts into an abstract description, which is then refined through logic synthesis to represent the final implemented hardware. Another language suitable for co-design is called SpecChart [6]. Development of hardware and software estimators for this language is carried out by researchers. Particularly, software estimation is based on a generic processor model, while hardware estimations are based on several area models [7,8]. These tools perform system partitioning at very abstract level in a first step, while evaluation is performed at low level in the second step. Hence, they are able to make very accurate estimations. However, they suffer from low simulation speed, which in turn slows down the convergence of finding a suitable system configuration.

Transaction level models (TLM) [9] have the advantages of decoupling functionality from communication between system blocks to meet short simulation times. TLMs are used for different purposes depending on the level of abstraction they are applied at [10]. In [11], TLM is used in a trace-based simulator, where processing times of software routines running on a processor or Central Processing Unit (CPU) are expressed as delay functions and memory reads/writes are mapped to bus latencies. This mechanism only allows for identification of hardware architecture timing bottlenecks without any indication on meeting absolute constraints. Moreover, this method overly abstracts software representation and hence does not account well for impacts from the software side. Ptolemy [12] is a design framework that targets the modelling, simulation and design of embedded systems by considering different models of computation, however, with the main focus on specification and code generation. Click [13] is an approach for specifying packet processing functionalities in an efficient way, however, without providing means for evaluation of their performance on specific system architectures.

In contrast to the above concepts, which either focus on software or hardware modelling, our methodology is based on a medium level of abstraction, where modelling of full software functionality is abstracted using SDL and the description of virtual hardware prototyping models is based on SystemC/C++.

3 The Design Methodology

The design problem of a parameterizable System-on-Chip (SoC) can be formulated as a search problem. As software and hardware are the two main components that build a whole SoC, each design point (d_n) can be defined as a combination of software and hardware parameters reflecting their architectural design variations. As an example, we assume that both hardware and software designs depend on two parameters each, and that each of these parameters has two possible assignments. Therefore, we will have in this case $2^4 = 16$ different design points forming a so called search space (S) as shown in Fig. 1. Notice that out of these points we have only six design points that meet our system's multi-objective functions, and hence they form a solution space (R).

Searching for design points belonging to the solution space in complicated SoCs cannot be done in an exhaustive manner due to an exponential number of design points. Moreover, estimating performance metrics for each configuration requires costly simulation and analysis of the system. Some evolutionary algorithms are used in [14] to reduce searching complexity. As finding an efficient heuristic is not the focus of our work, we use a simple heuristic that first considers the hardware dimension and then performs software variations only on promising hardware design points: those that are close or meet the system objective functions. This approach does not necessarily lead to an optimal solution, but to a local optimum that fulfils the given constraints.

The proposed flow for SDL/VP co-design is depicted in Fig. 2. Starting with system specifications and intended functionality, an initial partitioning of system tasks between hardware and software can be made. The design of software applications is carried out in SDL with an abstract style of modelling, from which C code is generated. SDL applications are then integrated together with a selected Real-Time Operating System (RTOS) and the required device drivers forming the whole software part of an embedded system. On the other hand, a VP representing the hardware system architecture can be constructed from off-the-shelf building blocks like processors and proprietary modules such as hardware accelerators, in addition to the memory sub-system, on-chip interconnects and

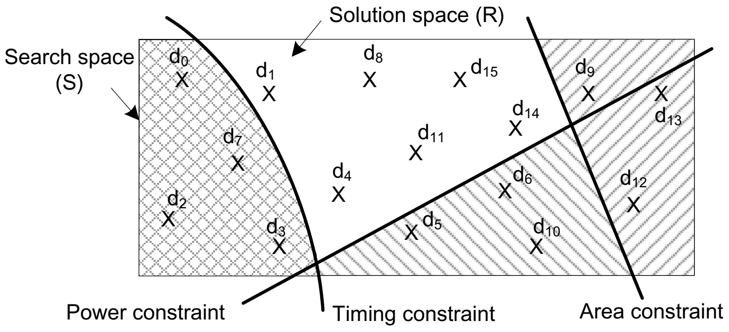


Fig. 1. System design problem viewed as a searching problem

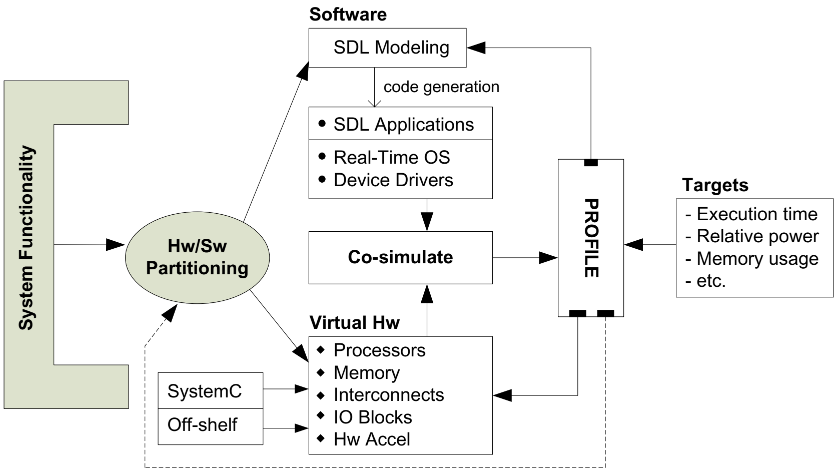


Fig. 2. Embedded system SDL/Virtual Prototype co-design flow

Input/Output blocks. The binary of the software component is then co-simulated with the developed VP. During simulation, profiling takes place by monitoring special system events based on selected target figures like software processing time, power consumption and memory usage. Analysis of simulated hardware and software design variations can then be performed by comparing the evaluation results against given system constraints. In case of the constraints are not met, further configurations are applied to hardware and/or software components before starting another iteration to evaluate the new system design point. Another possibility is to migrate some system tasks from software to hardware and vice versa to satisfy performance and area constraints, respectively.

The VP can be configured by selecting different hardware blocks to achieve a certain data or signal processing task. Thus, it can switch between candidates such as Digital Signal Processors (DSPs), Application Specific Instruction Set Processors (ASIPs) and general purpose processors. CPUs can also be customized by modifying their Arithmetic Logic Unit (ALU), pipeline architecture, cache associativity and size. The memory sub-system can also be investigated for various sizes, hierarchies and read/write latencies. The on-chip communication infrastructure can be configured to use different bus standards and arbitration schemes, or a network-on-chip topology. These configurations will help to evaluate the impact of hardware architecture on the metrics of interest. The design of different blocks in a VP is usually performed at relatively high abstraction levels based on SystemC/C++ which can be gradually refined as soon as the solution space is defined and reduced. This is especially required for area estimations which are dependent on transistor libraries and hence can be precisely determined only at low implementation levels.

Software partitions can be modelled in SDL in terms of communicating processes, where each process is specified as a set of interconnected abstract Finite

State Machines (FSM). In this way, we achieve a clear and abstract way of modelling software by focusing on program semantics rather than the language itself. Modification of software at this level is much faster than at lower levels. For instance, different implementations of the algorithm can change the functionality per process and its interaction with other processes. Other optimizations at this level could be reducing SDL process communication overhead and memory accesses, and distributing software tasks among SDL processes in order to exploit locality of data and instructions. SDL modelling might also be useful to cope with software challenges associated with next-generation multi-core embedded systems. For example, the number of software threads and the way they communicate and synchronize, and the degree of parallelism is highly influenced by the number of available SDL processes and their interconnections.

The proposed co-design flow employs combined software and hardware modelling at reasonable abstraction level, allowing the capturing of accurate information for system-level analysis and identification of design points that meet specified objective functions. Moreover, it provides means for Hw/Sw partitioning in modern SoCs as well as for optimization of both software and hardware within the same design flow. This approach also provides implicit verification of software and bridges the gap between software and hardware design methodologies. After finding a suitable design point, hardware refinement towards pure VHDL can be made for final chip tape-out and SDL generated code can be optionally replaced with pure C-code for efficiency considerations. As a use case, we will present in the following sections the hardware and software design of a simple mobile phone system and its architectural exploration based on the proposed methodology.

4 Virtual Hardware Platform

We build a VP of a mobile phone platform using tools from VaST Systems Technology Corporation [15]. This platform is intentionally designed to be based on a multi-core processor, which enables us to perform future investigations about parallelism opportunities for high data rate communication standards. Within the context of this paper, this platform is used as a single-core system, where only one of the available cores is activated.

The architecture of our platform is inspired from the RealView PB11MPCore baseboard provided by ARM [16]. A simplified block diagram of our VP is depicted in Fig. 3. Standard building blocks like processor and interconnects, are taken from the model library provided by the tool vendor. An ARM11 MPCore processor realizes four ARM11 cores, which are representative for state-of-the-art processors in mobile phones [17]. Each of these processors is equipped with an L1 data and instruction cache and a local timer. A snoop control unit is used to maintain the coherency between processors' local L1 data caches. A distributed interrupt controller (not shown in Fig. 3) is responsible for dispatching input interrupt lines into the corresponding cores. Each of the four cores features an eight-stage datapath pipeline and a Memory Management Unit (MMU) to

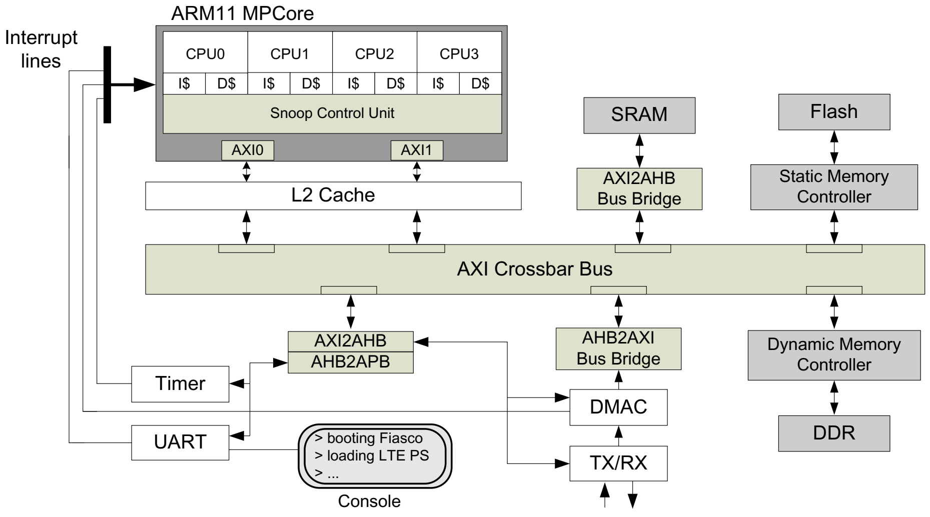


Fig. 3. Hardware architecture of mobile phone platform built in a Virtual Platform

support virtual memory. Moreover, the processor subsystem contains a L2 cache to improve the performance of data intensive applications.

The platform employs different kinds of memory blocks. Flash memory is used to hold the device firmware or boot code, which initializes the system : for example, configuring the L2 cache and memory controllers. The firmware then gives control to the operating system which runs from the RAM.

This Double Data Rate Synchronous Dynamic RAM (DDR) is used as an external main memory of the system. The operating system and applications running above are executed from this memory. The read and write latencies of this memory are adjusted according to the state-of-the-art mobile phone platform [18]. In addition, an internal SRAM memory offers low access latencies and hence can be used to store small and time-critical data and/or code. Static and dynamic memory controllers implement the interface protocol required for reading and writing data to Flash and DDR memories, respectively.

To off-load the processor, the Direct Memory Access Controller (DMAC) is used for performing efficient burst transfers from and to the IO blocks. The DMAC has an associated device driver software for configuration. The Transmit/Receive (TX/RX) is an IO block that implements the physical interface for transmitting and receiving data frames, for instance through an Ethernet port. Once a data frame is received, the TX/RX block signals the DMA requesting a data transfer to memory. After that, the DMA copies the data from the TX/RX block to a predefined location in memory. When the copy is completed, the DMA controller raises an interrupt notifying the processor that a radio frame exists in memory and is ready to be fetched and processed. Upon transmission, the processor triggers DMA transfers from memory to the TX/RX block by writing to its configuration register.

System components communicate and exchange data through a high speed crossbar bus. Other busses with different standards and speed rates are connected to the main bus using bus bridges. The bus bridge allows translation between distinct bus protocols. The AMBA High-performance Bus (AHB) is used for internal memory and the DMAC, whereas the Advanced Peripheral Bus (APB) is used to connect low speed peripherals, such as timer and UART. The timer can be used for the scheduling functionality of the operating system, however, in our case the core's internal timer is used for this purpose. User interaction is provided by a console connected to the UART.

5 Software Stack

The stack representing the software component of our mobile phone platform is made up of two parts. The Long Term Evolution (LTE) communication subsystem for decoding and processing of data packets, and the L4/Fiasco microkernel as an RTOS on top of which the protocol stack executes. These two software entities and their details are illustrated in the following subsections.

5.1 LTE Modem Application

The modem application in our case represents the layer 2 (L2) functionality of the LTE protocol stack. It is divided into uplink and downlink processing paths, representing the communication protocol from mobile phone to base station and vice versa, respectively. Both uplink and downlink consist of the Medium Access Control (MAC), the Radio Link Control (RLC), and the Packet Data Convergence Protocol (PDCP) sublayers [19]. As part of abstract software modelling, the SDL model of the modem application implements data plane sublayers in several concurrent processes communicating through signals. Control plane processing is not considered since it does not have much impact on the data processing time. Fig. 4 shows the processing tasks implemented in the SDL model.

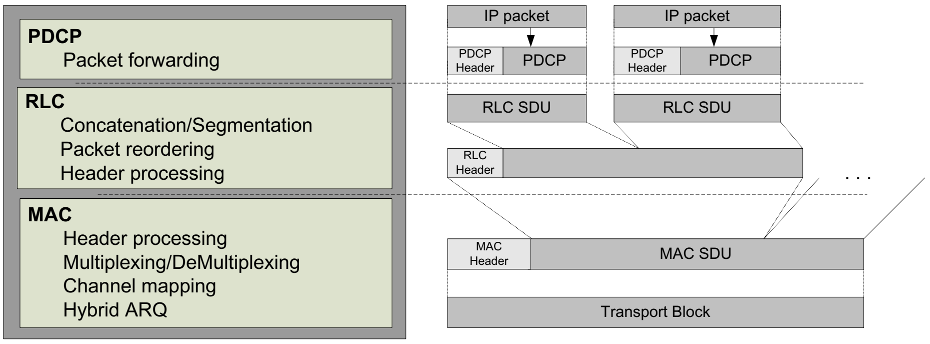
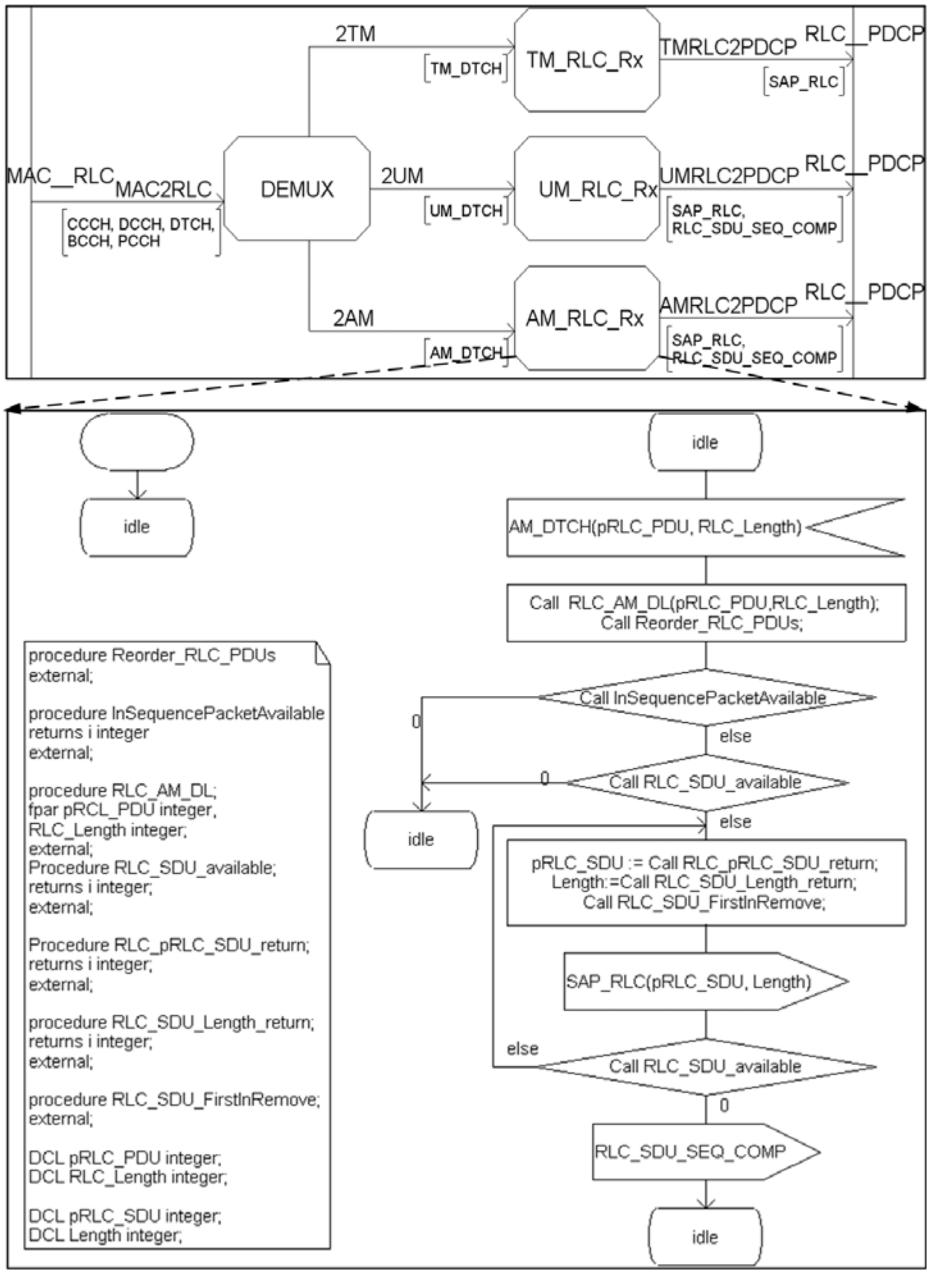


Fig. 4. Implemented functionality of the modem subsystem in SDL


2TM
TM_DTCH
TM_RLC_Rx
TMRLC2PDCP
RLC__PDCP
SAP_RLC
MAC__RLC
MAC2RLC
CCCH, DCCH, DTCH, BCCH, PCCH
DEMUX
2UM
UM_DTCH
UM_RLC_Rx
UMRLC2PDCP
RLC__PDCP
SAP_RLC, RLC_SDU_SEQ_COMP
2AM
AM_DTCH
AM_RLC_Rx
AMRLC2PDCP
RLC__PDCP
SAP_RLC, RLC_SDU_SEQ_COMP
idle
idle
AM_DTCH(pRLC_PDU, RLC_Length)
Call RLC_AM_DL(pRLC_PDU,RLC_Length); Call Reorder_RLC_PDUs;
Call InSequencePacketAvailable
0
else
Call RLC_SDU_available
0
else
idle
pRLC_SDU := Call RLC_pRLC_SDU_return; Length:=Call RLC_SDU_Length_return; Call RLC_SDU_FirstInRemove;
SAP_RLC(pRLC_SDU, Length)
else
Call RLC_SDU_available
0
RLC_SDU_SEQ_COMP
idle

```
procedure Reorder_RLC_PDUs
external;

procedure InSequencePacketAvailable
returns i integer
external;

procedure RLC_AM_DL;
fpar pRCL_PDU integer,
RLC_Length integer;
external;
Procedure RLC_SDU_available;
returns i integer;
external;

Procedure RLC_pRLC_SDU_return;
returns i integer;
external;

procedure RLC_SDU_Length_return;
returns i integer;
external;

procedure RLC_SDU_FirstInRemove;
external;

DCL pRLC_PDU integer;
DCL RLC_Length integer;

DCL pRLC_SDU integer;
DCL Length integer;
```

Fig. 5. The RLC downlink block architecture and its Acknowledged Mode process behavior

When a transport block is received at the mobile terminal, MAC processing starts by applying the Hybrid Automatic Repeat Request (HARQ) process, which retransmits transport blocks for error recovery. When a correct transport block is decoded, header processing starts by decoding the MAC header to extract data like logical channel Identification (LCID) and the Service Data Unit (SDU) length. Afterwards, the downlink shared transport channel is mapped into corresponding control and traffic logical channels, which realize the interface with the RLC sublayer. Consequently, it demultiplexes MAC SDUs into their corresponding logical channels. In the opposite direction, i.e. uplink processing, inverse operations are performed starting with multiplexing of MAC SDUs and ending with header generation to form the transport block ready for transmission.

Data processing continues at the RLC sub-layer after data is passed from the MAC sublayer. First, the RLC header is decoded and then segmentation processing is applied by unpacking an RLC Protocol Data Unit (PDU) into RLC SDUs, or portion of SDUs. This process depends also on the size of the packets. If the transport block is small, due to bad channel conditions, the RLC SDU may be split among several RLC PDUs. As out-of-order packets might be produced during handover, packet order is corrected in RLC by reordering the packets based on the sequence number carried out in the RLC header. These operations summarize the unacknowledged mode of RLC processing. Fig. 5 illustrates the architecture of the RLC entity in the downlink direction. This entity demultiplexes packet data into different modes of RLC processing implemented in three different processes. The communication of these processes with the rest of the system is performed using SDL messages via the SDL channels. As an example, the Acknowledged Mode process *AM_RLC_Rx* is described using an Extended Finite State Machine (EFM). For space reasons, the LTE SDL system is not fully demonstrated in this paper.

The PDCP sub-layer is implemented as packet forwarding in our investigations, but will be extended in the future enabling complete L2 modelling in SDL. According to LTE specifications, two transport blocks should be processed in 1 ms leading to a processing budget of 0,5 ms per transport block. Performance analysis of LTE protocol processing in [20] shows that 13 % of this time is occupied by MAC and RLC sublayer processing. This means the timing requirements for our architectural exploration in section 6 must be set to 65 μs.

SDL modelling and code generation of the previously described protocol stack functionality is performed using the IBM Rational tool, SDL Suite [21]. By setting the operating system interface option to POSIX, we are able to generate code capable to run on L4/Fiasco microkernel, which supports the same API standard. The size of the SDL kernel itself accounts for 11,000 lines of code and hence will impose some overhead on the generated SDL software system.

5.2 The L4/Fiasco Microkernel

The modem application in our mobile phone system runs on top of L4/Fiasco based operating system [22]. The latter is composed of two layers, the L4/Fiasco microkernel and the L4 runtime environment (L4Re) as shown in Fig. 6. The

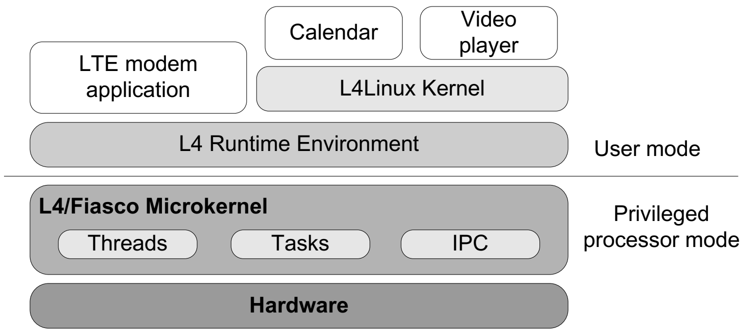


Fig. 6. The L4/Fiasco based operating system

selection of such a modern operating system which adopts the concept of microkernels is due to the fact that microkernels can act as robust RTOSs. For instance, microkernels aim at running only the most necessary functionality in the processor privileged mode. Hence, it requires smaller code size which reduces complexity and percentage of errors in the privileged mode. Moreover, microkernels offer good isolation characteristics by separating the communication subsystem from untrusted components like freeware applications. In addition, it supports virtualization by allowing the execution of general purpose operating systems like embedded Linux, where mobile applications like calendar and video codec can run, together with a proprietary RTOS responsible for executing the communication protocol in a mobile phone. The services of each layer of the deployed OS will be illustrated in the following.

The L4/Fiasco microkernel is the only component running in processor privileged mode and is responsible for managing the underlying hardware. Based on its nature, it provides a minimal set of mechanisms like tasks, threads, and Interprocess Communication (IPC). Fiasco kernel services are implemented in terms of kernel objects. A task comprises an address space where one or more threads can execute. Multiple threads within a task are scheduled by Fiasco's priority-based and preemptive scheduler. An IPC kernel object provides the basic communication mechanism in L4- based systems and is used mainly for transmitting arbitrary data between threads. On the other hand, the L4Re offers a basic set of abstractions and services, which are useful to implement user- level applications on top of the L4/Fiasco microkernel. It consists of a set of libraries mainly responsible for memory and IO resource management.

In this work, we consider only the SDL model of the protocol stack as an application to execute directly on top of the L4Re and the microkernel, which in turn provides real-time capabilities.

6 Architectural Exploration Case Study

As a case study we apply the proposed methodology for exploring the impact of different hardware and software architectural parameters on system performance.

The objective of this exploration is to analyze the mobile phone platform. So, the focus here is to see how the execution time of the protocol stack is influenced by different selected parameters. Particularly, the main goal of this case study is the customization of processor and memory subsystem in order to meet the timing constraint of 65 μs previously derived in subsection 5.1.

The parameters selected for system exploration are core frequency, memory latency, cache size, and the number of threads realized by our SDL model. These parameters have impact on system metrics. As the power of a CMOS-based chip grows linearly with the operating frequency, higher processor frequencies will result in more power consumption. On the other hand, this will improve the speed of our system as the processor will be able to handle more instructions within a fixed period of time, thus shortening the protocol stack execution time. Note that the frequency of a processor is not the only deciding factor for system performance. Actually the latter depends on the nature of the application being executed since the processor has always to communicate with memory. Therefore, it makes sense to investigate the impact of memory response or read/write latency on our objective function. This parameter representing memory throughput has also a linear impact on power consumption. Another important aspect is to find a suitable L1 cache size. Actually, large cache sizes should shorten the packet processing time by exploiting both spatial and temporal locality of program code and packet data. This however happens at the expense of increasing the chip area resulting in a higher fabrication cost. As an architectural software design parameter, we increase the thread density of our protocol stack SDL model. Threads have impact on systems memory and performance as each thread is assigned a dedicated stack in memory and has to be managed by the operating system. However, threads are also useful for exploiting concurrency and parallelism in a system as long as the application allows it. Although we know how these design parameters impact the system, it is still unclear how huge this impact is. This is going to be illustrated within the rest of this section. This study will also allow us to see the trade-off between different metrics, which contributes to making our final design decisions.

Figure 7 shows the profiling setup and demonstrates how the processing times are measured. The whole system is simulated with a cycle-approximate level of accuracy using CoMET from VaST. We run the generated code of the SDL-modelled protocol stack on top of the L4/Fiasco based operating system, which in turn executes on the designed hardware platform. In this study, IP packets' payload is generated in software. The data is then passed to the uplink data plane processing which outputs a valid transport block as it would be received from a base station via the air interface. After that, the downlink part of the stack is triggered to process the transport block before it signals the IP data generation process to start further iterations. In this way we are able to investigate both processing paths at the same time.

As our objective function is to meet the timing requirements with respect to the protocol stack execution time, timer tags are integrated into the stack's generated code enclosing both uplink and downlink processing paths. Timing

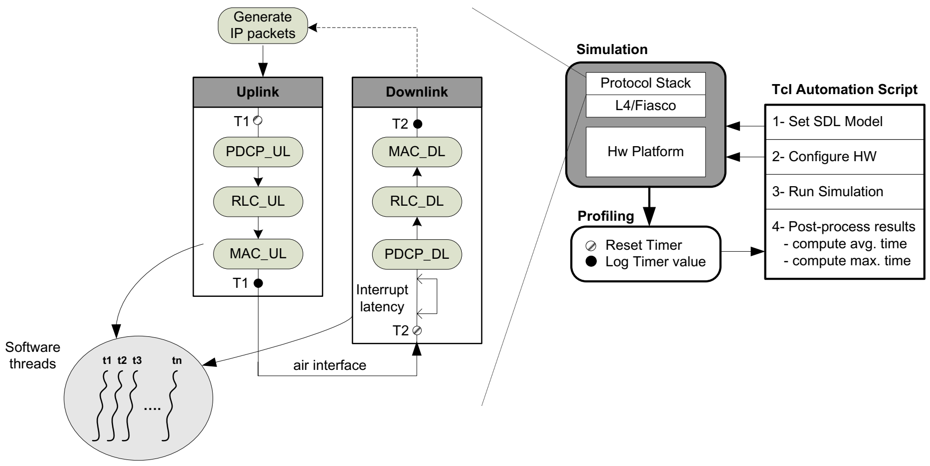
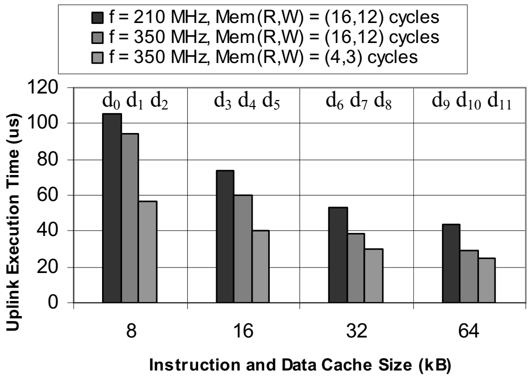


Fig. 7. Simulation and profiling setup for evaluation of several design configurations with respect to the execution time

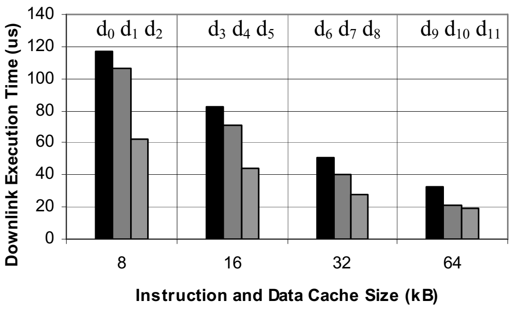
measurements are performed by waiting for the timer tags during simulation to start and stop corresponding timers leading to the measurements of the uplink and downlink processing times. A Tcl automation script controls the whole evaluation process which starts by selecting the SDL model which adheres to the software design parameter. Afterwards, it adapts the hardware architecture according to the assigned hardware parameter values and runs the simulation where the performance profiling is also made. The evaluation ends by post- processing the recorded timer values. Since different packet processing iterations have different processing times, mainly due to different hardware states, we collect the processing time for hundreds of iterations and post-process them to compute the average and maximum processing time per packet.

Having four design parameters each with wide range of values will result in a huge number of combinations or design points. To reduce the number of design points, we select parameter values within the range, which is acceptable for embedded systems. For instance, the core frequency can be assigned to two possible values 210 MHz and 350 MHz. However, the (read,write) memory latency parameters are assigned to (16,12) or (4,3) cycles at a reference frequency of 100 MHz. For the cache size, we consider four distinct values (8, 16, 32, and 64 kbytes) which are realistic candidates in a typical mobile device. For the software parameter, we design SDL models with different number of running threads (1, 2, 6, and 17) by distributing the functionality of the MAC and RLC sublayers into a higher number of SDL processes. In this use case, we are not concerned with efficient mapping of SDL processes into concurrent worker threads. This analysis will be carried out later on when dealing with multicore architectures in order to explore and exploit parallelism in the modem application.

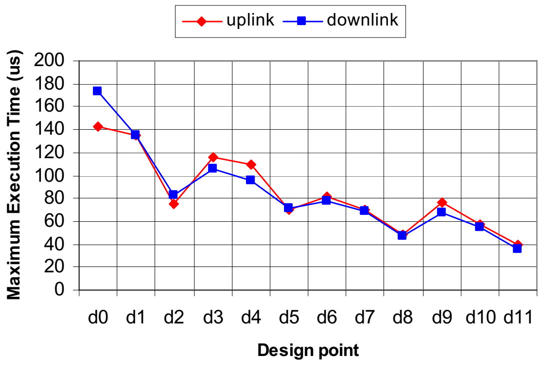
We apply the heuristic of considering design points with hardware variations under stable software conditions. Particularly, we set the number of threads to



(a)



(b)



(c)

Fig. 8. Evaluation of several design points through variation of hardware parameters. Average execution time is shown in subplots (a) and (b), whereas plot (c) shows the maximum execution time of both uplink and downlink processing.

one and vary all other hardware parameters. As a second step, we vary software parameters only on the set of design points showing good results from the first evaluation step. The uplink and downlink processing times corresponding to design points d_0 to d_{15} considered in the first evaluation step are depicted in Fig. 8. In this figure the design points are grouped according to their cache size.

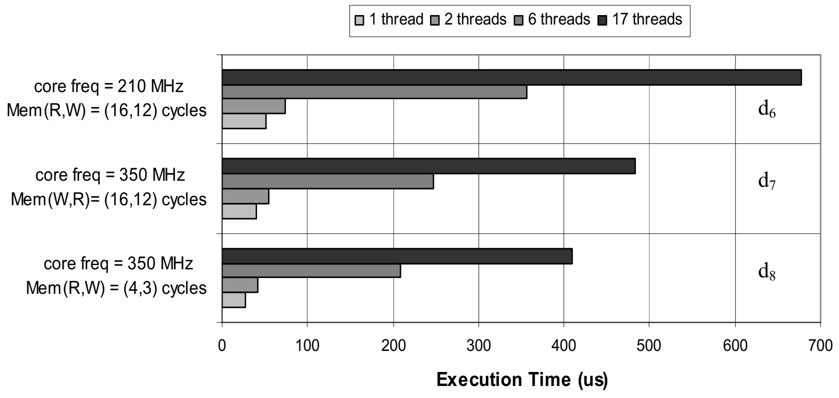


Fig. 9. Evaluation of different SDL threaded models at 32 kB cache size

From the results, we can notice that the achieved processing times are shortened with higher data caches reaching 44 μs for uplink and 33 μs for downlink at core frequency of 210 MHz and memory (read,write) latency of (16,12) cycles. By increasing the processing frequency to 350 MHz, the execution time of both uplink and downlink is reduced by 10 % and 35 % at a cache size of 8 kB and 64 kB, respectively. This shows that for small caches with large number of misses, the system performance is limited by the memory response rather than processor speed. This is justified in the third bar where the memory latency is reduced to 25 %, leading to 40 % reduction in processing time at 8 kB cache size, and only 10 % reduction at 16 kB. In plot (c), the maximum execution time per design configuration point is depicted. Design points with a cache size of 32 kB and 64 kB are the only configurations which fulfil or almost fulfil the timing constraints. On the other hand, a small gain of 9 μs can be observed by doubling the cache size from 32 kB to 64 kB under the same frequency and memory latency. This shows that design points with a 64 kB cache size are not worth to consider due to their area overhead in comparison with the performance gain they can bring. As a result, only design points based on 32 kB cache, i.e. $d_6 - d_8$, are considered in the next evaluation step.

As a second step, we apply different variations of threaded SDL models into design points $d_6 - d_8$. As Fig. 9 shows, the downlink execution time increases by 45 % with configuration of two threads, and much larger with 6 and 17 threads. It is obvious that we will not gain performance with higher number of threads due to thread management overhead without possibility for parallelism since the protocol stack is running on one core. However, we realize that the impact of thread management is quite huge in a single core and this has to do with the overhead of thread management coming from the SDL kernel as well as the L4/Fiasco based operating system. This architectural exploration and analysis allows us to highlight this issue, which should be taken into consideration for further optimization especially when executing the stack on several cores.

Finally, we select the design point d_7, where the processor with a 32 kB cache operates at 350 MHz and the memory (read,write) latency is (16,12) cycles at 100 MHz. In addition, the single threaded SDL model is selected. This configuration achieves an average execution time which meets the timing constraints and is on average 80 % faster than other design configurations (see Fig. 8). Moreover, it provides a good balance between area and power costs.

7 Conclusion

In this paper we propose a hardware/software co-design flow for embedded systems. Within this flow, software components are abstractly modelled using SDL, while hardware components are emulated in software using the concept of virtual prototyping. This approach allows for fast and early investigations of several design options for both hardware and software due to the low effort and high speed associated with such modelling techniques. To prove our methodology, an architectural exploration of a mobile phone platform is considered. We demonstrate an SDL-modelled LTE protocol stack, an L4/Fiasco based RTOS, and a designed virtual prototype of a mobile terminal. We customize the processor and memory subsystem of the platform by rapidly obtaining a suitable design configuration which meets the required timing constraints and provides 80 % packet processing speedup compared to other unoptimized implementations. In addition, the achieved design parameters provide a balanced power/area consumption tradeoff. For further study, different Hw/Sw partitioning configurations can be applied and evaluated in a similar way. As a future work, we will adapt the software part to make efficient utilization of multi-core architectures and make further investigations about the performance gain, power consumption and scalability of such a multi-core based communication system.

Acknowledgement. The authors acknowledge the excellent cooperation with all partners within the ICT-eMuCo project and the support by the European Commission. Further information is available on the project web-site: `http://www.emuco.eu`.

References

1. Grötker, T., Liao, S., Martin, G., Swan, S.: System Design with SystemC. Kluwer Academic Publishers, Boston (2002)
2. Thiele, L., Wandeler, E.: Performance Analysis of Distributed Embedded Systems. In: Zurawski, R. (ed.) Embedded Systems Handbook, CRC Press, Boca Raton (2005)
3. Ernst, R., Henkel, J., Benner, T.: Hardware-Software Cosynthesis for Microcontrollers. IEEE Design & Test of Computers 10(4), 64–75 (1993)
4. Thomas, D.E., Adams, J.K., Schmit, H.: A Model and Methodology for Hardware-Software Codesign. IEEE Design & Test of Computers 10(3), 6–15 (1993)

5. Chiodo, M., Giusto, P., Jurecska, A., Hsieh, H.C., Vincentelli, A.S., Lavagno, L.: Hardware-Software Codesign of Embedded Systems. IEEE Micro. 14(4), 26–36 (1994)
6. Gajski, D.D., Vahid, F.: Specification and Design of Embedded Hardware-Software Systems. IEEE Design & Test of Computers 12(1), 53–67 (1995)
7. Gong, J., Gajski, D.D., Narayan, S.: Software Estimation using a Generic-Processor Model. In: Proceedings of the 1995 European Conference on Design and Test, p. 498. IEEE Computer Society, Washington, DC (1995)
8. Vahid, F., Gajski, D.D.: Specification Partitioning for System Design. In: Proceedings of the 29th ACM/IEEE Design Automation Conference, pp. 219–224. IEEE Computer Society Press, Los Alamitos (1992)
9. Cai, L., Gajski, D.D.: Transaction Level Modeling: An Overview, http://www.cecs.uci.edu/conference_proceedings/isss_2003/cai_transaction.pdf
10. Donlin, A.: Transaction Level Modeling: Flows and Use Models. In: Hardware/Software Codesign and System Synthesis, 2004. CODES + ISSS 2004, pp. 75–80. ACM, New York (2004)
11. Wild, T., Herkersdorf, A., Ohlendorf, R.: Performance Evaluation for System-on-Chip Architectures using Trace-based Transaction Level Simulation. In: Proceedings of the Conference on Design, Automation and Test in Europe, pp. 248–253. European Design and Automation Association, Leuven (2006)
12. Buck, J., Ha, S., Lee, E.A., Messerschmitt, D.G.: Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems, http://ptolemy.eecs.berkeley.edu/publications/papers/94/JEurSim/JEurSim.pdf
13. Kohler, E., Morris, R., Chen, B., Jannotti, J., Kaashoek, F.M.: The Click Modular Router. ACM Trans. on Computer Systems 18(3), 263–297 (2000)
14. Palesi, M.: Multi-Objective Design Space Exploration using Genetic Algorithms. In: Proceedings of the Tenth International Symposium on Hardware/Software Codesign, CODES 2002. ACM, New York (2002)
15. The VaST Systems Technology Corporation, http://www.vastsystems.com
16. RealView Platform Baseboard for the ARM11 MPCore, http://www.arm.com/products/DevTools/PB11MPCore.html
17. Silven, O., Jyrkkä, K.: Observations on Power-Efficiency Trends in Mobile Communication Devices. EURASIP Journal on Embedded Systems (2007)
18. Hessel, S., Bruns, F., Bilgic, A., Lackorzynski, A., Härtig, H., Hausner, J.: Acceleration of the L4/Fiasco Microkernel Using Scratchpad Memory. In: International Workshop on Virtualization in Mobile Computing, MobiVirt 2008. ACM, New York (2008)
19. Evolved Universal Terrestrial Radio Access (E-UTRA), 3GPP Specifications: Rel8 (December 2008), http://www.3gpp.org
20. Szczesny, D., Showk, A., Hessel, S., Hildebrand, U., Frascolla, V., Bilgic, A.: Performance Analysis of LTE Protocol Processing on an ARM based Mobile Platform. Accepted for 11th International Symposium on System-on-Chip (SoC 2009), Tampere, Finland (October 2009)
21. IBM® Rational® SDL Suite™, http://www.ibm.com/software/awdtools/sdlsuite/
22. The Fiasco Microkernel, http://os.inf.tu-dresden.de/fiasco

Author Index