Lea Kutvonen
Nancy Alonistioti (Eds.)

LNCS 3543

Distributed Applications and Interoperable Systems

5th IFIP WG 6.1 International Conference, DAIS 2005
Athens, Greece, June 2005
Proceedings

ifip

Springer

Lea Kutvonen   Nancy Alonistioti (Eds.)

Distributed Applications and Interoperable Systems

5th IFIP WG 6.1 International Conference, DAIS 2005
Athens, Greece, June 15-17, 2005
Proceedings

Volume Editors

Lea Kutvonen
University of Helsinki
Department of Computer Science
P.O. Box 68, Gustaf Hällströmin katu 2b, 00014 Helsinki, Finland
E-mail: Lea.Kutvonen@cs.Helsinki.FI

Nancy Alonistioti
University of Athens
Department of Informatics and Telecommunications
157 84 Panepistimiopolis, Ilissia, Athens, Greece
E-mail: nancy@di.uoa.gr

Library of Congress Control Number: 2005926896

CR Subject Classification (1998): D.2, C.2.4, I.2.11, D.4, H.4

ISSN 0302-9743
ISBN-10 3-540-26262-8 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-26262-6 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11498094 06/3142 5 4 3 2 1 0

Preface

This volume contains the proceedings of the IFIP WG 6.1 International Working Conference on Distributed Applications and Interoperable Systems V held in Athens, Greece, on June 15–17, 2005.

The conference program presented the state of the art in research concerning distributed and interoperable systems. The emergence of 4th-generation communication systems, the evolution of Internet technologies, the convergence of telecom and datacom, wireless and fixed communication systems and applications pave the path for ubiquitous service and application provision. Innovative solutions are required for the development, implementation and operation of distributed applications in complex IT environments full of diversity and heterogeneity. Today, the emerging wide spectrum of distributed systems – ranging from ambient intelligence to global computing scenarios – lacks systematic application development support. Following the evolution of the field, DAIS 2005 focuses on models, technologies and platforms for interoperable, scalable and adaptable distributed applications within all kinds of computing environments.

The papers presented at DAIS 2005 cover methodological aspects of building and architecting distributed and interoperable services, interoperability technologies, context- and location-based applications, configurability of communication services, performance issues, data-integration issues, and Web services. In comparison to earlier events, the submissions showed increased interest towards methodological aspects and large-scale system interoperability.

These proceedings contain 16 regular and 5 short papers, which were selected in a careful, international reviewing process. The DAIS 2005 conference was sponsored by IFIP (International Federation for Information Processing) and it was the fifth conference in the DAIS series of events organized by IFIP Working Group 6.1. The previous conferences in this series took place in Cottbus, Germany (1997), Helsinki, Finland (1999), Krakow, Poland (2001), and Paris, France (2003). In Paris, DAIS was organized for the first time in conjunction with the FMOODS conference (Formal Methods and Open Object-Based Distributed Systems), and that practice was followed in Athens as well.

The conference program was complemented with joint invited talks with the FMOODS audience. Prof. Gordon Blair (Lancaster University) discussed *Middleware and the Divergent Grid*, while Prof. Rocco de Nicola (University of Florence) addressed *Programming and Reasoning on Global Computers with Evolving Topologies*. Prof. Andreas Reuter (European Media Laboratory GmbH) shed light on the area of *Application Integration with Emphasis on Orchestration and Consistency*.

Finally, we would like to take this opportunity to thank the numerous people whose work made this conference possible. We wish to thank the University of Athens for hosting the event, and Prof. Lazaros Merakos for acting as a general chair of the joint DAIS and FMOODS event. Costas Polychronopoulos provided the Web services for us. Prof. Hartmut König took care of the publicity for the event. The Steering Committee with Profs. Guy Leduc, Hartmut König, Kurt Geihs, and Elie Najm extended their helping hand on many essential occasions.

It is the good technical papers that make a conference. Especially we thank the submitters and reviewers for their contributions towards an interesting conference.

June 2005 Nancy Alonistioti and Lea Kutvonen

Conference Committees

Chairs

Steering Committee	Guy Leduc, University of Liège, Belgium; chair 2004
	Elie Najm, ENST, Paris, France; chair 2005
	Hartmut König, BTU Cottbus, Germany
	Kurt Geihs, University of Kassel, Germany
	Lea Kutvonen, University of Helsinki, Finland; member in 2005
General Chair	Lazaros Merakos, University of Athens, Greece
Program Co-chairs	Nancy Alonistioti, University of Athens, Greece
	Lea Kutvonen, University of Helsinki, Finland
Publicity Chair	Hartmut König, BTU Cottbus, Germany
Organizing Chair	Costas Polychronopoulos, University of Athens, Greece

Program Committee

M. Ahamed, Georgia Tech, USA
M. Anagnostou, National Technical University of Athens, Greece
S. Baker, IONA, Ireland
Y. Berbers, Katholieke Universiteit Leuven, Belgium
A. Beugnard, ENST-Bretagne, France
D. Bourse, Motorola Labs, France
I. Demeure, ENST, France
F. Eliassen, University of Oslo, Norway
K. Geihs, University of Kassel, Germany
S. Gessler, NEC Europe, Germany
D. Hagimont, INRIA, France
S. Hallsteinsen, Sintef, Norway
P. Honeyman, University of Michigan, USA
J. Indulska, University of Queensland, Australia
E. Jul, DIKU, Denmark
A. Kaloxylos, University of the Peloponnese, Greece
V. Karamcheti, New York University, USA
H. König, BTU Cottbus, Germany
H. Krumm, University of Dortmund, Germany
W. Lamersdorf, University of Hamburg, Germany
C. Linnhoff-Popien, University of Munich, Germany
P. Merle, INRIA, France
E. Najm, ENST, France

G. Papadopoulos, University of Cyprus, Cyprus
K. Raymond, DSTC, Australia
D. Schmidt, University of California, USA
W. Schreiner, Johannes Kepler University Linz, Austria
T. Senivongse, Chulalongkorn University, Thailand
J.-B. Stefani, INRIA, France
M. Virvou, University of Piraeus, Greece
D. Zeghlache, INT, France
K. Zielinski, UMM Krakow, Poland

External Referees

T. Alanko, University of Helsinki, Finland
T. Niklander, University of Helsinki, Finland
S. Pan-ngum, Chulalongkorn University, Thailand
C. Phongphanphanee, Chulalongkorn University, Thailand
S. Srinutapong, Microsoft, Thailand
Y. Teng-amnuay, Chulalongkorn University, Thailand
K. Raatikainen, University of Helsinki, Finland
S. Ruohomaa, University of Helsinki, Finland
T. Ruokolainen, University of Helsinki, Finland

Table of Contents

Towards Real-Time Middleware for Applications of Vehicular Ad Hoc Networks

René Meier, Barbara Hughes, Raymond Cunningham, and Vinny Cahill

Distributed Systems Group, Department of Computer Science,
Trinity College Dublin, Ireland
{rene.meier, barbara.hughes, raymond.cunningham, vinny.cahill}@cs.tcd.ie

Abstract. Applications of inter-vehicle and vehicle-to-roadside communication that make use of vehicular ad hoc networks (VANETs) will often require reliable communication that provides guaranteed real-time message propagation. This paper describes an event-based middleware, called RT-STEAM, designed to meet these requirements. Unlike other event systems, RT-STEAM does not rely on a centralized event broker or look-up service while still supporting event channels providing hard real-time event delivery. RT-STEAM event filtering can be based on subject, content and/or proximity. Proximity filters define geographical areas within which events are delivered. To guarantee real-time communication, we exploit proximity-based event propagation to guarantee real-time constraints within the defined proximities only. The proximity within which real-time guarantees are available is adapted to maintain time bounds while allowing changes to membership and topology as is typical of VANETs. This Space-Elastic Model of real-time communication is the first to directly address adaptation in the space domain to guarantee real-time constraints.

1 Introduction

Many Ad hoc wireless networks comprise sets of mobile nodes connected by wireless links that form arbitrary wireless network topologies without the use of any centralized access point or infrastructure. Ad hoc wireless networks are inherently self-creating, self-organizing and self-administering [1].

While most research in ad hoc networks has assumed a random waypoint mobility model in a network of a particular shape, e.g., rectangular, [2], the specific patterns of vehicle movement make inter-vehicle and vehicle-to-roadside networks distinctive. In particular, the potential for high speeds and the limited dimensionality afforded by a confined roadway, differentiates vehicular ad hoc networks (VANETs) from other ad hoc networks.

By enabling inter-vehicle and vehicle-to-roadside communication a broad range of applications in the areas of cooperative driver assistance and mobile information systems are being developed. One of the more sophisticated applications for inter-vehicle communication is the platooning of vehicles [3]. For example, the lead vehicle in such an application may broadcast sensor information to coordinate movement and potentially reduce the consumption of fuel.

A possible application of vehicle-to-roadside communication is in next generation urban traffic control (UTC) systems. A vehicle (or set of vehicles) approaching a junction could inform a traffic light controller at the junction of its pending arrival. The traffic light controller could then change the traffic light sequence to allow the approaching vehicle to pass through the junction without stopping. When a number of vehicles are approaching the junction, and if the traffic light controller is informed of the presence of the approaching vehicles, then the controller could optimize the traffic flow across the junction. This time constrained communication between vehicles and traffic light controllers should continue reliably during peak times when a large number of vehicles approach the junction from a number of different directions. The potential for contention increases as the number of vehicles communicating with the controller increases.

Both the vehicle platooning and UTC applications require a communication paradigm that supports dynamic changes to the topology of the underlying wireless network, for example to accommodate vehicles joining and leaving a platoon, as well as delivery guarantees for time-critical messages. In the UTC application scenario described above, the traffic light controller needs to be informed of the presence of the approaching vehicle in sufficient time to allow it to change the flow of traffic across the junction.

This paper describes an event-based middleware, called RT-STEAM, designed for ad hoc networks. Unlike other event systems, RT-STEAM does not rely on centralized services while still supporting event channels providing hard real-time event delivery.

RT-STEAM is based on an implicit event model [4] and has been designed for mobile applications and wireless ad hoc networks. RT-STEAM differs from other event services in that it does not rely on the presence of any centralized components, such as event brokers or look-up services, and supports distributed techniques for identifying and delivering events of interest based on location. RT-STEAM supports decentralised approaches for discovering peers, for routing events using a distributed naming scheme, and for event filtering based on combining multiple filters. Filters may be applied to the subject and the content of events, and may be used to define geographical areas within which events are valid. Such proximity-based filtering represents a natural way to filter events of interest in mobile applications.

RT-STEAM provides a programming model based on the concept of event channels. A number of event channel classes with different temporal and reliability attributes are available to integrate real-time support into the event channel model. Depending on the guarantees available from the underlying network, the proximity characteristically associated with an event channel may require adaptation to maintain the required real-time guarantees while allowing changes to membership and topology as is typical of VANETs. This Space-Elastic model is the first to directly address adaptation in the space domain to maintain real-time guarantees.

An underlying assumption of the Space-Elastic model is that a real-time application in a VANET can specify and interpret specified bounds in space (the proximity) where real-time guarantees are critical. This assumption relates to the observations in [3], that the relevance of data to a specific geographical area is one of the key features of an inter-vehicle network and that mission-critical (e.g., emergency braking notification), and non-critical (e.g., weather reports), communication competes for the limited available resources. Thus, Space-Elastic applications must be space-aware, i.e., operate correctly in a dynamic proximity, and information-sensitive, i.e., aware of the criticality of different sources of information (event channels). In the vehicle platooning scenario, the proximity where real-time communication is critical may bound the platoon and, for example, vehicles within the vicinity of the platoon, moving in the same direction. In the UTC scenario described, the critical proximity may be the minimum area within which the controller has sufficient time to change the flow of traffic following communication with an approaching vehicle.

The reminder of this paper is structured as follows: Section 2 introduces RT-STEAM's programming model and architecture. Section 3 presents our Space-Elastic Model of real-time communication and describes its approach to exploiting proximity-based event propagation for maintaining time bounds. Section 4 outlines RT-STEAM's communication architecture. Finally, section 5 concludes this paper and outlines our future work.

2 RT-STEAM

The design of the RT-STEAM architecture is motivated by the hypothesis that there are applications in which mobile components are more likely to interact once they are in close proximity. For example, a vehicle is interested in receiving emergency braking notifications from other vehicles only when these vehicles are within close proximity. Similarly, traffic light controllers are only interested in arrival notifications from vehicles that are located within a certain range of their own locations. This means that the closer event consumers are located to a producer the more likely they are to be interested in the events that it produces. Significantly, this implies that *events are relevant within a certain geographical area surrounding a producer*.

Event Types, Proximities, and Channels. RT-STEAM implements an implicit event model [4] that allows event producers to publish events of a specific *event type* and consumers to subscribe to events of particular event types. Producers may publish events of several event types and consumers may subscribe to one or more event types**.**

To facilitate the kind of location-aware application described above, RT-STEAM supports a programming model that allows producers to bound the area within which their events are relevant and to define Quality of Service (QoS) attributes describing the real-time constraints of these events. Such a combination of event type, geographical area and QoS is called an *event channel*. Producers *announce* event channels, i.e., they announce the type of event they intend to *raise* together with the geographical area, called the *proximity*, within which events of this type are to be disseminated with the required QoS constraints. Thus, an event channel announcement bounds event propagation to a defined proximity where required QoS constraints are guaranteed. Events are delivered to consumers only if they reside within a proximity where the QoS constraints for the event type are satisfied.

Producers may define proximities independently of their physical transmission range with an underlying group communication system routing event messages from producer to consumer using a multi-hop protocol. Proximities may be of arbitrary shape and may be defined as nested and overlapping areas. Nesting allows a large proximity to contain a smaller proximity subdividing the large area. Fig. 1 depicts two overlapping proximities of different shape and illustrates that multiple consuming and producing entities may reside inside a proximity. These proximities have been associated with events of *type A* and *type B* as well as *QoS A* and *QoS B* respectively. Consequently, consumers handling these event types receive events if they reside inside the appropriate proximity. An example of overlapping proximities might include a vehicle disseminating an emergency braking notification within the vicinity of a traffic light controller that is also receiving arrival notifications from approaching vehicles.

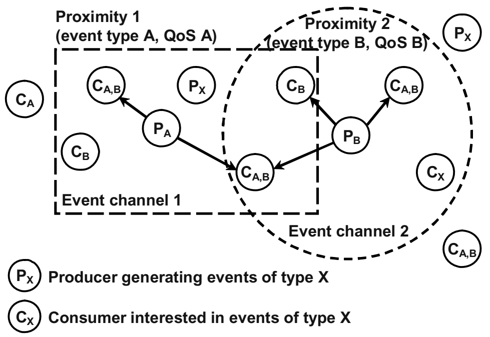


Fig. 1. Disseminating events using event types, proximities and event channels

Supporting Mobility. RT-STEAM has been designed to support applications in which application components can be either stationary or mobile and interact based on their geographical location. This implies that the RT-STEAM middleware as well as the entities hosted by a particular machine are aware of their geographical location at any given time. RT-STEAM includes a location service that uses sensor data to compute the current geographical location of its host machine and entities. To suit outdoor applications, for example, in the traffic management domain, RT-STEAM exploits a version of the location service that uses a GPS satellite receiver to provide latitude and longitude coordinates.

In addition to supporting stationary and mobile entities RT-STEAM allows proximities to be either stationary or mobile. A stationary proximity is attached to a fixed point in space whereas a mobile proximity is mapped to a moving position represented by the location of a specific mobile producer. Hence, a mobile proximity moves with the location of the producer to which it has been attached. This implies that mobile consumers and producers may be moving within a mobile proximity. For example, a group of platooning vehicles might interact using a proximity that has been defined by the leading vehicle. Such a proximity might be attached to the position of the leader moving with its location.

Subscribing to Event Types. Consumers must subscribe to event types in order to have the middleware deliver subsequent events to them when located inside any proximity where events of this type are raised, until they unsubscribe. A consumer may move from one proximity to another without re-issuing a subscription when entering the new proximity. Thus, subscriptions are persistent and will be applied transparently by the middleware every time a subscriber enters a new proximity. This implies that a subscription to a specific event type applies to all proximities handling these events even though the subscriber may only receive a subset of these events at any time. A single subscription may result in events of a particular event type raised by different producers in multiple proximities and with different QoS being delivered over time. Hence, the set of events received by a subscriber at a certain time depends on its movements as well as on the movements of producers and proximities.

Defining Event Channels. Applications define event channels to specify the functional and non-functional attributes of the events they intend to disseminate. A RT-STEAM event channel is defined by a *subject* and a set of attributes, i.e., its event type, as well as of a proximity description and QoS constraints representing its non-functional attributes. The subject defines the name of a specific event type and the content defines the names and types of a set of associated parameters. RT-STEAM event instances are defined in a similar manner by specifying a subject and content parameter values. Producers and consumers must use a common vocabulary defined by the application to agree on the name of an event type. An event type and an event instance that have the same subject must have an identical content structure, i.e., the set of parameter names and types must be consistent. As described in more detail below, distributed event filters may be applied to the subject, content, and proximity attribute of an event.

Applying Event Notification Filters. RT-STEAM supports three different classes of event filter, namely *subject filters*, *content filters*, and *proximity filters*. These filters may be combined and a particular event is only delivered to a consumer if all filters match. Subject filters match the subject of events allowing a consumer to specify the event type in which it is interested. Content filters contain a filter expression that can be matched against the values of an event's parameters. Content filters are specified using filter expressions describing the constraints of a specific consumer. These filter expressions may contain equality, magnitude and range operators as well as ordering relations. They may include variable, consumer local information. Proximity filters are location filters that define the geographic scope within which events are relevant and correspond to the proximity attribute associated with an event type.

Locating Proximities. Instead of requiring a naming service to locate entities that wish to interact, RT-STEAM provides a discovery service that uses beacons to discover proximities. Once a proximity has been discovered, the associated events will be delivered to subscribers that are located inside the proximity. This service is also responsible for mapping discovered proximities to subscriptions and to the underlying *proximity-based communication groups* [5]. Hence, it causes the middleware to join a proximity-based multicast group of interest, i.e., for which it has either a subscription or an announcement, once the host machine is within the associated proximity-bound and to leave the proximity group upon departure from the area.

RT-STEAM's Distributed Naming Scheme. There are two essential issues that need to be addressed when mapping announcements and subscriptions to proximity groups. Firstly, a naming scheme for uniquely identifying groups is required and secondly, a means for consumers and producers to obtain the correct group identifiers needs to be provided. These issues are traditionally addressed by employing approaches based either on statically generating a fixed number of unique and well-known group identifiers or on using a (centralized) lookup service for generating and retrieving group identifiers. However, neither of these approaches suffices for applications that use VANETs due to their dynamic nature and their inherently distributed infrastructure.

The RT-STEAM event model exploits a decentralized naming scheme in which identifiers representing groups can be computed from event channel descriptions. Each combination of event type, proximity, and QoS is considered to be unique throughout a system assuming that there is no justification for applications to define multiple identical event channels. A textual description of a subject, proximity, and QoS triple is used as input to a hashing algorithm to dynamically generate hash keys that represent identifiers using local rather than global knowledge. Upon discovery of a proximity and the associated event type and QoS, consumers compute the corresponding group identifier if the subject is of interest. This naming scheme allows consumers to subsequently use these identifiers to join groups in which relevant events are disseminated. Moreover, it allows consumers that are not interested in certain events to avoid joining irrelevant groups and consequently, from receiving unwanted events even though they might reside inside the proximity associated with a group.

3 The Space-Elastic Model

The Hard real-time event communication in a highly dynamic VANET is challenging. We scope the problem by exploiting the proximity filters defined by RT-STEAM to reduce the area of a VANET where real-time communication is required to within the defined proximity bounds only. However, the dynamics of a VANET also impacts the real-time guarantees available within the proximity bound. For example, in the vehicle platooning scenario, movement through city environments, with increased obstacles such as tall buildings, may impact the ability to guarantee real-time communication within the entire proximity defined. In this case, we dynamically adapt the proximity bound to maintain the required real-time guarantees. This dynamic proximity, or space-elastic, adaptation is at the core of our Space-Elastic model.

Challenges to Real-Time Communication. Prior to introducing the Space-Elastic model we present an overview of the challenges to hard real-time communication in a dynamic VANET. The challenges to MANETs discussed in [6], e.g., dynamic connectivity, unpredictable latency and limited resource availability are exacerbated in VANETs, for the following reasons.

Limited Communication Duration. In [2], an analytical investigation of topological dynamics based on classical vehicular traffic theory [7], presents valuable information about the relationship between velocity and communication duration. It is shown that in the case of oncoming traffic, and a high average velocity of 130km/h the probability of maintaining connectivity for a communication duration of less than 30s is less than 0.1. Furthermore, as investigated in [3], the transmission range of the radio system significantly influences these duration figures, and that for radio systems supporting a distance less than 500m, the communication duration must be reduced to below 10s. The highly dynamic communication duration expected in a high velocity

VANET, directly impacts, and must be reflected, in the design of medium-access and routing protocols.

Diverse Range of Information. Combining safety, e.g. emergency braking notification, with comfort, e.g. traffic jam notification, in VANETs, leads to a diversity of information criticality and thus required communication guarantees. This diversity influences resource allocation and routing, e.g., prioritized bandwidth usage for emergency notification, and must be reflected in policies available to the middleware protocols.

Impact of Traffic Density on Multi-Hop Connectivity. Both the vehicle platooning and UTC scenarios may require multi-hop connectivity to coordinate driver assistance. As shown in [3], traffic dynamics, in terms of the velocity of vehicles and the density of traffic, impacts the maximum number of hops available for information exchange. For example, 5 hop communication is available with a probability of more than 90% only if the radio system supports at least a range of 1km, and the traffic volume is high. In low-density traffic volumes, with the same radio support, the probability of achieving 5 hops falls significantly to below 20%. To reduce the impact of limited multi-hop connectivity on real-time communication, we investigate dynamically adapting the geographical area within which real-time guarantees are available, and thus within which route discovery is necessary. This is an example of space-elastic adaptation which we discuss in the following sections.

Limitations of Time-Elastic Adaptation. Given the challenges outlined, achieving hard real-time guarantees in a VANET is potentially impossible without restricting the characteristics of the real-time applications, the environment or both. Traditional hard real-time systems have restricted the application, for example, by assuming a known upper bound on the number of participants or assuming known connectivity to intermediate components [8, 9]. These static assumptions are not applicable in a dynamic environment.

The Timely Computing Base (TCB) [10], addresses the problem of achieving synchrony in dynamic environments with uncertain timeliness. In [10], Veríssimo and Casimiro introduce the idea that the probability of achieving real-time communication in dynamic environments changes over time. This observation motivated the specification of the time-elastic class of applications [11]. Time-elastic applications are the subset of real-time applications where it is more beneficial to execute in a degraded mode (e.g., extended time latencies), than to stop, or more critically, have unexpected failures due to a change in the assumed coverage. Inter-vehicle and vehicle-to-roadside applications that require hard real-time guarantees are not time-elastic. Thus the TCB model and time-elastic classification are not applicable. To achieve hard real-time communication in a dynamic VANET, we look instead at the proximity, or space, within which the guarantees are required.

Space-Elastic Adaptation. The timeliness properties of hard real-time applications are invariant. Adapting the terminology of [10], hard real-time applications require guarantees that P(within T from t(e)) = 1, i.e., the probability that a time-dependent execution will complete within (a time interval) T from (start time) t(e), is guaranteed for all timeliness properties. A model of real-time communication must observe this definition of timeliness properties.

In the Space-Elastic model, we look at the space, or proximity, within which real-time properties are required, which is related to the observation in [3], that one of the key features in an inter-vehicle network is the relevance of data to a particular geographical area. For example, in emergency braking notification, the critical proximity where real-time guarantees are required, is within a bound geographical area of the braking vehicle. Beyond this proximity, a "safe distance" exists, e.g., determined by the speed limitations of the road and standard braking distances of vehicles [12], where the notification is not critical and real-time guarantees do not apply. The Space-Elastic Model exploits this rationale to guarantee real-time constraints within specific proximity-bounds only.

Guaranteed hard real-time communication within a specified proximity only, requires a reduction in the scope of the timeliness property specified previously. Assuming space(E, size) is a function that bounds a geographical space of a defined size in relation to entity E, (a mobile or stationary node in a VANET), the timeliness property becomes P(within T from t(e) over space (E, size) only) = 1.

In Fig. 2, the circle represents the proximity-bound for real-time guarantees. The entity, E in this case, may represent a traffic light, in the UTC example, or a mobile vehicle, where the proximity-bound is associated with, and moves with, the vehicle and represents the critical proximity for real-time communication with this vehicle.

The Space-Elastic model assumes that real-time applications are space-aware. In addition, given the potential for increased network topology dynamics, the limited communication duration, and influence of traffic density on multi-hop route discovery, a further assumption of the Space-Elastic model is that defined proximity bounds are adaptable, e.g., reducing the size, or changing the shape, to guarantee desired real-time communication, regardless of the dynamics of the VANET within the proximity. Thus, Space-elastic real-time applications must tolerate dynamic proximity-bound adaptation. For example in the UTC example, an emergency vehicle (e.g., an ambulance) moving towards a traffic light may have a desired proximity within which real-time communication with the roadside is guaranteed. However, given the challenges to communication in VANETs previously discussed, real-time communication within the desired proximity may not be possible, and a reduced proximity, representing the minimum geographical area, within the desired proximity, where real-time communication is currently achievable, is available only. In this case, it is assumed that the space-elastic application knows the minimum proximity where real-time guarantees are critical and adaptation between the maximum (desired) and minimum (critical) proximity is tolerated. Failure to guarantee real-time communication in the critical proximity is a real-time

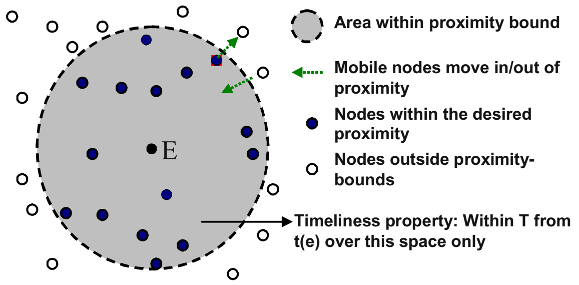

E
Area within proximity bound
Mobile nodes move in/out of proximity
Nodes within the desired proximity
Nodes outside proximity-bounds
Timeliness property: Within T from t(e) over this space only

Fig. 2. Timeliness properties within a proximity-bound only

failure, the consequences of which and the actions to arise are determined by the characteristics of the real-time class. For example, failure to guarantee hard real-time properties in the *critical* region is a critical failure, and transition to a fail-safe or fail-operational [13] state may be possible.

Fig. 3 illustrates proximity adaptation, in this case motivated by route failures in the desired proximity rendering real-time communication no longer possible. In this example, real-time guarantees are available in the area highlighted as the adapted proximity-bound only.

For a space-elastic application to benefit from proximity adaptation the behavior of the space-elastic application must also be adaptable. For example, in vehicle platooning, a desired proximity for inter-vehicle real-time communication for vehicle coordination, may span driving lanes used by vehicles moving in the opposite direction, e.g. for oncoming accident notification [14], to the platoon. In this case, although the desired proximity may contribute rich context about traffic conditions to the platoon, the critical proximity for real-time inter-vehicle communication, bounds the platoon and vehicles within a vicinity of the platoon, e.g. determined by the speed of the platoon, driving in the same direction only. Thus, proximity (or space) adaptation to encompass the critical proximity only is advantageous. In addition, given the reduction in the proximity where real-time guarantees are achievable, the platoon may adapt behavior accordingly, e.g., reducing speed, until such a time when real-time communication within the original desired proximity is achievable.

Space-elastic (Sε) real-time applications are defined as follows: Given an application A, represented by a set of properties P_A, A belongs to the space-elastic class iff none of the duration properties (T) derived from any property P of A, require an invariant space.

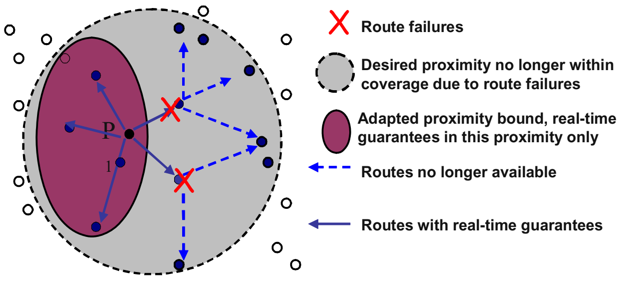

Route failures
Desired proximity no longer within coverage due to route failures
Adapted proximity bound, real-time guarantees in this proximity only
Routes no longer available
Routes with real-time guarantees
P
1

Fig. 3. Adaptive space-elasticity

$\forall A \in S\varepsilon, \forall P \in P_A$: space(A, T) is not invariant

In practical terms, space-elastic applications (A) are those that can accept a bounded proximity (space(A, T)) within which specific real-time application properties (T) are guaranteed. Space-elastic applications must be able to execute correctly in an adaptable proximity or space.

Finally, it is interesting to investigate the role of space-elastic adaptation in a partitioned VANET, i.e., where a subset of nodes (a stationary or mobile participant in a VANET), are no longer contactable i.e., within one-hop or multi-hop connectivity range, for example, due to buildings in a city environment. A partitioned VANET adversely impacts real-time communication, e.g. established routes may no longer be available, new routes may not be discovered due to inaccessible (partitioned) nodes and resources previously reserved for real-time communication, may no longer be available. Furthermore, real-time communication spanning the desired proximity may no longer be possible. Using the space-elastic model, the proximity where real-time properties are guaranteed is adapted to reflect the partition. Real-time communication is guaranteed in a reduced local proximity (at a minimum the critical proximity) only, until such time as a merge occurs with other proximities in the desired proximity.

Fig. 4 (a) illustrates the desired proximity for real-time communication, by an entity, represented by E. Initially real-time event communication is guaranteed within the desired proximity. Fig. 4 (b) represents the network topology after an interval, *T*, where, for example, vehicle movement has caused the VANET, and thus the desired proximity, to partition. The desired proximity has been adapted (Proximity$_b$), to the area where real-time guarantees are available only. Also illustrated, is the scenario where other local proximities (i.e., Proximity$_c$), possibly with different real-time constraints, co-exist within the original desired proximity bounds.

A space-elastic real-time application specifies both desired and critical proximity-bounds coupled with associated real-time guarantees, by combin ing RT-STEAM proximity filters with associated event channels, yielding real-time event-based communication in a defined proximity-bound only. To achieve real-time guarantees, the communication architecture used by RT-STEAM combines a proactive, mobility-aware routing and resource reservation protocol, at the network layer, with a predictable time-bounded medium-access control protocol. We outline this communication architecture in the following section.

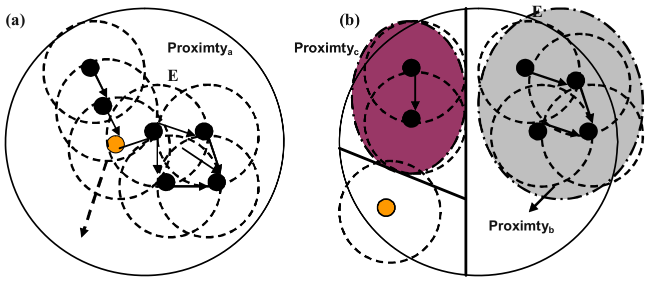


Fig. 4. Dynamic proximity adaptation in a partitioned network

4 Real-Time Communication Architecture

We have completed an implementation and evaluation of RT-STEAM, including proximity-bounds specification, distributed naming etc. for non real-time channels [15] In this section, we present our work in progress on medium-access control and routing protocols to extend our implementation to include real-time channels.

Our medium access control protocol, TBMAC [16] provides, with high probability, time bounded access to the wireless medium to mobile hosts in a multi-hop ad hoc network. The TBMAC protocol is based on time division multiple access with dynamic but predictable slot allocation. TBMAC uses a lightweight atomic multicast protocol to achieve distributed agreement on slot allocation.

To reduce the probability of contention between mobile hosts, the geographical area occupied by the mobile hosts is statically divided into a number of cells in a similar approach to [17]. Each cell is allocated a particular radio channel to use. Each mobile host is required to know its location (using GPS) and from this which cell it is in and what radio channel to use to communicate with other mobile hosts in the cell.

In a similar way to the IEEE 802.11 standard, the TBMAC protocol divides access to the wireless medium within a cell into two distinct time periods:

- Contention Free Period (CFP)
- Contention Period (CP)

Both the CFP and the CP are divided into a set of slots. A CFP followed by a CP constitute a round of the TBMAC protocol. Dividing access to the medium into these two well-known time periods requires the clocks of all the mobile hosts in the network to be synchronized (e.g., each host using a GPS receiver [18]). Once a mobile host has been allocated a CFP slot, it has predictable access to the wireless medium until it leaves the cell or fails. Mobile hosts that do not have allocated CFP slots contend with each other to request CFP slots to be allocated to them in the CP.

The motivation for our routing protocol is to combine proactive routing with mobility-awareness in order to reduce the unpredictability of a dynamic VANET. Our routing and resource reservation protocol (PMRR) attempts to discover and maintain real-time constrained routes within a proximity-bound, e.g., as specified by the Space-Elastic Model. Given the observations in [2], that in “normal” traffic scenarios, the probability of a stable topology varies between 90 and ~100%, and that the topology remains absolutely stable for up to 60s in cases of lower densities and lower relative velocities, our use of proactive routing and resource reservation to guarantee critical communication, appears justified in a VANET.

PMRR executes in two phases. The route discovery phase attempts to proactively discover real-time constrained routes and reserve resources within a defined proximity, providing timely failure notification if routes or resources are not available. The route maintenance phase uses mobility and link quality prediction [19] to assist in proactive route maintenance decisions, e.g., route repair prior to a route break occurring.

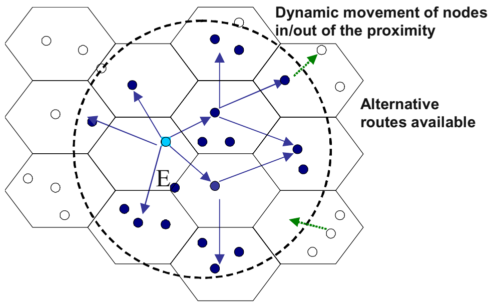

Dynamic movement of nodes in/out of the proximity
Alternative routes available
E

Fig. 5. Proactive route discovery and maintenance

The PMRR maintains one-hop neighbor routing information in routing tables at each hop, e.g., similar to AODV [17], to reduce routing overhead. Furthermore, the routing tables store all alternative routes (i.e., those routes where real-time constraints are guaranteed) from a node, i.e., either the originator or an intermediate node, thus reducing route maintenance latency. Similar to the work of Chen et al. [18], the PMRR utilizes selective probing. However, what is distinctive in this case is that traditional shortest path metrics are substituted by required real-time guarantees.

Fig. 5 illustrates the discovery of proactive routes for real-time communication with E, within a defined proximity-bound. E initially transmits in their allocated slot in the current cell. Using inter-cell slot allocation [16], the transmission is forwarded to neighbouring cells satisfying required real-time constraints only. This process continues for all cells in the proximity-bound, or until a cell is reached where the real-time constraints are no longer guaranteed, e.g., an empty cell.

Depending on the dynamics of the environment within the proximity and the space-elasticity of the application, proximity adaptation may be performed if real-time guarantees cannot be maintained within the entire proximity.

5 Conclusions

We have proposed our middleware for providing real-time communication in VANETs. An implementation and evaluation of our non real-time event-based middleware has been performed. We are currently implementing with the intent of evaluating our real-time middleware. We have currently developed a simulation of TBMAC, a distributed clock synchronisation protocol and a real-time driver for wireless communication using 802.11b.

Acknowledgements. The work described in this paper was partly supported by the Irish Higher Education Authority's Programme for Research in Third Level Institutions cycle 0 (1998-2001) and by the FET Programme of the Commission of the European Union under research contract IST-2000-26031 (CORTEX).

References

[1] S. Chakrabarti and A. Mishra, "QoS Issues in Ad Hoc Wireless Networks," *IEEE Communications Magazine*, vol. 39, pp. 142-148, 2001.

[2] M. Rudack, M. Meincke, and M. Lott, "On the Dynamics of Ad Hoc Networks for Inter Vehicle Communications (IVC)," in *Proceedings of the International Conference on Wireless Networks (ICWN02)*. Las Vegas, USA, 2002.

[3] H. Hartenstein, B. Bochow, A. Ebner, M. Lott, M. Radimirsch, and D. Vollmer, "Position-Aware Ad Hoc Wireless Networks for Inter-Vehicle Communications: The Fleetnet Project," in *Proceedings of the 2nd ACM International Symposium on Mobile Ad Hoc Networking & Computing (MobiHoc)*. Long Beach, CA, USA, 2001, pp. 259-262.

[4] R. Meier and V. Cahill, "Taxonomy of Distributed Event-Based Programming Systems," in *Proceedings of the International Workshop on Distributed Event-Based Systems (ICDCS/DEBS'02)*. Vienna, Austria: IEEE Computer Society, 2002, pp. 585-588.

[5] M. O. Killijian, R. Cunningham, R. Meier, L. Mazare, and V. Cahill, "Towards Group Communication for Mobile Participants," in *Proceedings of ACM Workshop on Principles of Mobile Computing (POMC'2001)*. Newport, Rhode Island, USA, 2001, pp. 75-82.

[6] B. Hughes and V. Cahill, "Towards Real-time Event-based Communication in Mobile Ad Hoc Wireless Networks," in *Proceedings of 2nd International Workshop on Real-Time LANS in the Internet Age 2003 (ECRTS/RTLIA03)*. Porto, Portugal, 2003, pp. 77-80.

[7] W. Leutzbach, "Introduction to the Theory of Traffic Flow," *Springer-Verlag*, 1988.

[8] J. Kaiser and M. Mock, "Implementing the Real-Time Publisher/Subscriber Model on the Controller Area Network (CAN)," in *Proceedings of the 2nd International Symposium on Object-oriented Real-time distributed Computing (ISORC99)*. Saint-Malo, France, 1999, pp. 172-181.

[9] Object Management Group, *CORBAservices: Common Object Services Specification - Event Service Specification*: Object Management Group, 1995.

[10] P. Verissimo and A. Casimiro, "The Timely Computing Base Model and Architecture," *IEEE Transactions on Computers - Special Issue on Asynchronous Real-Time Systems*, vol. 51, pp. 916-930, 2002.

[11] A. Casimiro and P. Verissimo, "Using the Timely Computing Base for Dependable QoS Adaptation," in *Proceedings of the 20th IEEE Symposium on Reliable Distributed Systems*. New Orleans, USA, 2001, pp. 208-217.

[12] U.S. Department of Transportation, "Amendment for Brake Performance," in Motor Carrier Safety Improvement Act 1999, Federal Motor Carrier Safety Association (FMCSA), 2002.

[13] H. Kopetz and P. Verissimo, *Real-time and Dependability Concepts*, 2nd ed: ACM-Press, Addision-Wesley, 1993.

[14] L. Briesemeister and G. Hommel, "Role-based Multicast in Highly Mobile but Sparsely Connected Ad Hoc Networks," in *Proceedings of Mobihoc 2000*. Boston, USA, 2000.

[15] R. Meier and V. Cahill, "Exploiting Proximity in Event-Based Middleware for Collaborative Mobile Applications," in *Proceedings of the 4th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS'03)*, *LNCS 2893*. Paris, France: Springer-Verlag, 2003, pp. 285-296.

[16] R. Cunningham and V. Cahill, "Time Bounded Medium Access Control for Ad Hoc Networks," in *Proceedings of the Second ACM International Workshop on Principles of Mobile Computing (POMC'02)*. Toulouse, France: ACM Press, 2002, pp. 1-8.

[17] C. E. Perkins, E. M. Belding-Royer, and S. R. Das, "Ad Hoc On-Demand Distance Vector (AODV) Routing," IETF Internet Draft June 2002.

[18] S. Chen and K. Nahrstedt, "Distributed Quaility-of-Service Routing in High-Speed Networks Based on Selective Probing," in *Proceedings of the 23rd Conference on Local Computer Networks (LCN98)*. Boston, USA: IEEE Computer Society, 1998.

[19] G. Gaertner and V. Cahill, "Understanding Link Quality in 802.11 Mobile Ad Hoc Networks," *IEEE Internet Computing*, vol. 8, pp. 55-60, 2004.

Adaptive Context Management Using a Component-Based Approach

Davy Preuveneers and Yolande Berbers

Department of Computer Science, K.U. Leuven Celestijnenlaan 200A,
B-3001 Leuven, Belgium
{davy.preuveneers, yolande.berbers}@cs.kuleuven.ac.be
http://www.cs.kuleuven.ac.be

Abstract. Context-awareness has become a necessity for adaptable intelligent applications and services. It is crucial for ubiquitous and pervasive computing because the context of a user or device serves as the source of information to adapt services. In this paper, we propose a modular context management system that is able to collect, transform, reason on and use context information to adapt services. By employing a component-based approach, we enable our infrastructure not only to support context-aware adaptation of services, but also to support adaptation of the context management system itself at deployment time and at runtime. This self-adaptation is based upon the service requirements and the current context of the device, such as the current resource usage or other devices in the neighborhood, resulting in an adaptive context management system for improved quality of service.

1 Introduction

Context-awareness [1] is considered to be the key challenge for making mobile devices aware of the situation of their users and their environment. This research area focuses on the management of context information in pervasive computing environments [2] where people are surrounded by and interacting with many unobtrusive networked devices. These devices offer personalized assistance by adapting their applications' intended functionalities to the current context of the user and the device. This context information includes current location and time, users' activities and preferences, devices' capabilities, and any information that may characterize user-service interactions. Context representation has evolved from simple key-value pairs [3] to more complex ontology models [4, 5, 6, 7] to provide semantical uniformity and universal interchangeability.

The context management system is the heart of a context-aware architecture and processes instantiations of this context model. It is responsible for information retrieval and dissemination, structured storage of context, transformation of and reasoning on information, and the decision process to initiate certain actions. The current trend towards context-aware architectures is explained by the growing need for applications and services that are more sensitive to user requirements but less dependent on user attention. Hence, a critical success factor of a context-aware architecture in a mobile and ubiquitous computing environment is the support available to adapt services to a broad range of hardware, such as PDAs, mobile phones and smartphones. The underlying context management system must support flexibility as well. Otherwise, the context management system can consume all resources and reduce the quality of service to the user. For this reason, we have used *components* as modular building blocks for the design and deployment of the adaptable services and for the underlying context management system.

A component-based development approach [8] is an ideal software engineering methodology for having flexible adaptation capabilities within applications to optimize resource usage and to adapt to changing working conditions. Advantages of using components include the possibility of live updating with better suited components [9], negotiating and enforcing resource contracts [10], distributing the execution and relocating component-based applications [11].

In section 2 we give a general overview of adaptation in our context-aware architecture. In section 3 we describe how our context management system fits within a component-based service-oriented platform. In section 4 we discuss our component-based implementation and support for self-adaptation. The modular composition manages the retrieval and dissemination of, the storage of, the reasoning on and the transformation of context information. In section 5 we evaluate our system and discuss future work. Section 6 provides an overview of related work. We end with conclusions in section 7.

2 Adaptation in a Context-Aware Architecture

Both the services and the context management system are subject to adaptation triggered by a changing context, as discussed in the following subsections.

2.1 Service Adaptation

First of all, service adaptation before deployment of a service ensures that the service is tailored to the capabilities of the device [12]. Secondly, service adaptation can also be activated during the execution of a service. By way of example, consider a video conferencing service that adapts to a reduced network bandwidth by lowering the video frame rate or by disabling video altogether. Our context management system initiates both deployment time and runtime adaptations by providing all the necessary information to activate the adaptations. Each service specifies constraints that define working conditions that guarantee proper execution of (a subset of) the provided functions of the service. These constraints are encapsulated by triggers. A context change causing certain constraints to be violated will then trigger the runtime adaptation of the service. A brief overview of the component-based service model is given in section 3.

2.2 Context Management Self-adaptation

Our context management system itself is also subject to adaptation. Specific components of the context management system can be eliminated if they are of no use to any service. For example, a due shortage of storage capacity triggers the context management to delete irrelevant context information to increase allocation space. If context-aware service adaptation only depends on the current value of local sensors, then the context storage can be eliminated altogether.

To not overcomplicate the self-adaptation of our context management system, the adaptation is only triggered by changes in resources and service requirements. This information is usually readily available and requires no intensive processing.

3 Context-Awareness Within a Component-Based Service Platform

A context-aware service platform requires the interaction between a context managing infrastructure and the services which offer personalized assistance through context-based adaptation. In this section we briefly introduce our component-based services and their interaction with the context management system.

In several computer science domains a service refers to a computational entity that offers a particular functionality to a possibly networked environment. Context-aware services in mobile computing also require support for user personalization, deployment on embedded systems, user mobility and service relocation. To accomplish this, we apply a component-based development methodology. A general overview of our component-based service is shown in Figure 1.

Components [8] provide the functional building blocks of a service and use *Component Ports* as communication gateways to other components. *Connectors* serve as the message channel between these ports. *Contracts* [10] define restrictions or requirements on two or more components or ports. They are used, for example, to limit or guarantee memory and network bandwidth availability or to define timing constraints in order to guarantee a certain quality of service. The *Context Control Block* is responsible for managing the context information. This *Context Control Block* is largely shared by all services on the same device to eliminate the need for duplication for all services. Only components with a service-specific function cannot be shared, such as those processing service-

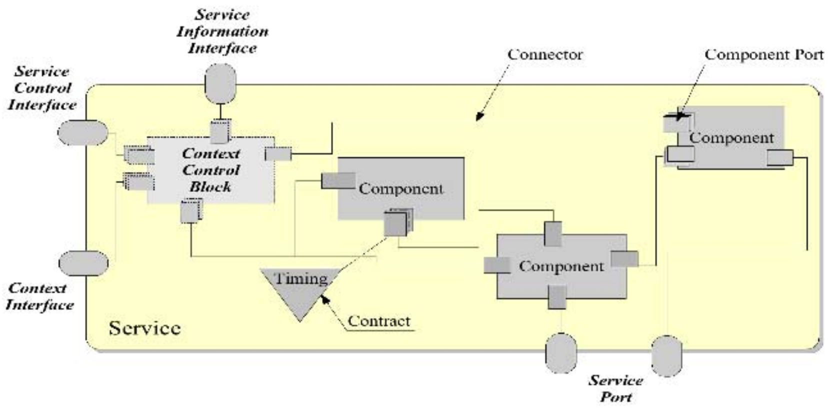


Fig. 1. Building blocks of a component-based service

specific required accuracy of information. The *Context Control Block* which is the focus of the rest of this paper is discussed in section 4.

A service is a wrapper around these entities with *Service Ports* as message gateway proxies to internal *Component Ports* and three management interfaces: the *Service Information Interface* to provide static semantic and syntactic information about a service during service discovery, the *Service Control Interface* to manage the launching, the relocating, the stopping and the uninstalling of a service, and the *Context Interface* for the service-specific context interchange and interaction with the *Context Control Block*, i.e. the context management system. As shown in section 4, the *Context Control Block* in itself is also composed of several subcomponents, each with a specific function.

4 Context Management

Retrieving and using context information require a uniform and interchangeable context representation. In the Context Toolkit [3], context is modeled as a set of key-value pairs. The more structured approaches for modeling context that have been proposed in the past use RDF [13], UAProf and CC/PP [14], and CSCP [15]. Ontologies, which allow the definition of more complex context models, have been used in several context modeling approaches [4, 5, 6]. For use in our context management system, we have designed a context ontology [7] based on the concepts of *User*, *Platform*, *Service* and *Environment*. This ontology is specifically targeted at context-driven adaptation of mobile services [16].

In the following subsections we discuss how an adaptable component-based context infrastructure, i.e. the *Context Control Block* as previously mentioned in section 3, is able to manage context information. Apart from managing context, the strength of our component-based approach relies on the fact that components with similar function but different runtime requirements can adapt the behavior of the context management system. The following subsections treat respectively a general overview of the context management system, context retrieval, context storage and context manipulation. Where applicable, they discuss how alternative and optional components can adapt the context management system to better suite the needs of the context-aware services.

4.1 General Overview of the Context Management System

The job of context management is performed by the *Context Control Block*. It consists of three components, each with a specific duty: *Context Retrieval*, *Context Storage* and *Context Manipulation*. See Figure 2 for a general overview.

4.2 Context Retrieval

This component gathers information from sensors or other providers on the system itself or in the neighborhood. Several issues with respect to the source of information and accuracy are discussed in the following subsections.

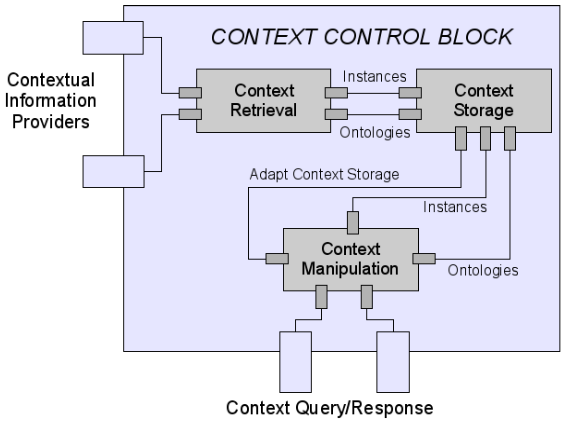


Fig. 2. Overview of the *Context Control Block* component

Sources of Information. We distinguish the following information providers:

Sensors: Information can be acquired by sensors attached to the device [17]. This low-level information is prone to measurement errors and can require transformation into conceptually richer information before being usable.

User profiling: Another source of information is acquired through user profiling. Based upon a history of previous user actions and input, a general profile with user preferences can be determined. It is clear that this kind of information is error prone and subject to change.

Third parties: Information can also be exchanged with other parties in the neighborhood. This information can be raw sensor data or be derived by combining all kinds of information.

Properties of Information. The value of information is determined not only by the information itself, but also by several information properties.

Accuracy: With sensors as information providers, it is easy to determine the real value of sensed data, as the granularity of measurement and accuracy is usually provided by the manufacturer of the equipment. This is not guaranteed for user profiled information. By combining information, small errors can propagate through the derivation chain and result in unusable information.

Reliability: Trust is important when a device is using information provided by third parties. Well-known devices have already had many occasions to prove their information to be accurate, whereas unknown devices have not had such an opportunity. Trust in other devices is managed by comparing all answers which influences trust in a positive or negative way. Note that this is no guarantee against hostile or malicious information providers.

Age: Information, which proved to be valuable before, can now be too old to be useful. Therefore, information is labeled with a time stamp defining its age. If the measuring or deriving of information takes too long, we can fall back on a previous value, but only if that information is not too old.

Component-Based Information Retrieval Modeling. An overview of all components involved in context retrieval is given in Figure 3. We have several components acting as information providers: *Sensors*, a *User Profile* and *Third Parties*. The *Information Requester* is the initiator of all information requests. In general, it monitors information that triggers service adaptations, such as changes in current network bandwidth. It sends these requests to the *Relevance Filter*, which forwards them to the information providers. Another function of the *Relevance Filter* component is to filter out unwanted information which has been pushed into the context management system by third parties. When several sources provide similar information in response to a request, the *Accuracy Comparator* selects the 'most reliable and accurate information' and forwards it back to the *Information Requester*. In Figure 3, a *Clock* also periodically sends a time signal and pushes this information to a *Timer* component. The *Timer* uses this information to enable configurable periodic signals. The *Information Requester* can then send a request to the *Timer* to be periodically notified to allow interval-based information monitoring.

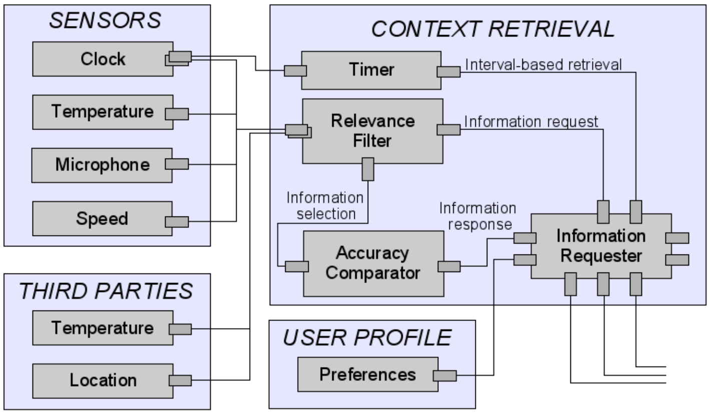


Fig. 3. Overview of the *Context Retrieval* component

Support for Adaptation. Depending on the processing capabilities of the device, components can be reduced in complexity or even eliminated. For example, instead of comparing and selecting on accuracy and reliability, we can replace the *Accuracy Comparator* by another component that only retains the first answer in a set of responses, with a possible reduction in accuracy of context information as a result. In the event a service only relies on the sensors of the device as information providers, then the *Relevance Filter* and *Accuracy Comparator* in Figure 3 can be completely removed, which means that the sensors are connected directly to the *Information Requester*.

4.3 Context Storing

A context repository ensures persistency of context information. It should save only up-to-date and relevant information in a way that queries and information updates can be handled efficiently without losing the semantics of the data.

Context Representation. Context representation involves two aspects: the information itself, for example '`Age=23`', and how it relates to other concepts, for example a person. The first aspect can be represented by a set of key-value attributes. The second determines the semantic meaning of this information.

Our context management system stores context as an expandable knowledge base of ontologies for semantic modeling of concepts and as a fact container with key-value pairs providing instantiations of these concepts. Using a separate attribute container simplifies querying the instantiations.

History of Context Information. For monitoring services it is useful to not only save the most recent value of a certain attribute, but to retain previously received values as well. In this way, the history of information can be exploited, for example by calculating the distance traveled by tracking the current location. This is implemented by including a time stamp with each key-value attribute.

Managing Outdated and Redundant Information. As storage capacity is limited, not all information can be retained. The oldest information is purged first, but the duration of relevance is not the same for all concepts. Therefore, we provide for each concept a lifetime during which it is still of value. If the information is older than the given lifetime, it is garbage collected.

Redundancy is a more complicated problem of information overhead. Should information be removed after it has been used to derive new information or should it be retained for later use? Our solution stores for each fact the latest occasion of when and how often it has been used. Rarely used and old information are the first candidates for removal. However, storing extra properties about facts requires storage space as well, and thus the advantages have to be thoroughly considered before implementing the removal of old or redundant data.

Component-Based Context Repository Modeling. The component-based repository is implemented as two different container components. See Figure 4 for

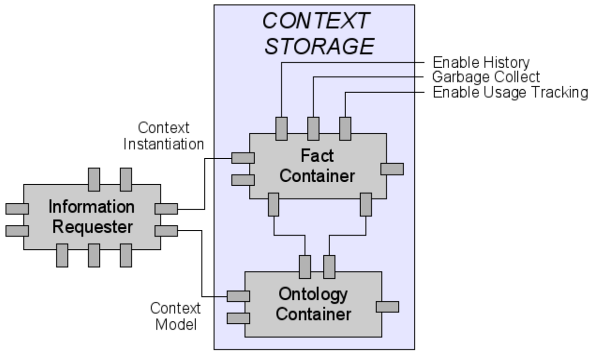


Fig. 4. Overview of the *Context Storage* component

Table 1. Example of an instantiation of the *Fact Container*

ID	Attribute	Value	Concept ID	Time Stamp	Last Used	Usage Count
1	Name	John	ID74358	07:53am	11:52am	7
2	Age	53	ID69230	07:54am	10:16am	2
3	Location	50°52' N 4°22' E	ID38031	02:37pm	02:37pm	119
4	Bandwidth	1112 kbps	ID16789	02:38pm	02:41pm	37
5	*LIFETIME*	30 sec	ID16789	-	-	-

an overview. The *Information Requester* sends new facts to the *Fact Container*, which holds instances of concepts from context ontologies. Two switches are used to enable and disable history preservation and usage tracking. When low on storage capacity, another signal is used to trigger the garbage collection of old facts. If one of the supplemental ontologies (i.e. not our base ontology [7], which serves as a common ground), is no longer referred to by a key-value pair, then it can be removed from the *Ontology Container* as well. An instantiation of the *Fact Container* is given in Table 1. Properties with respect to accuracy and reliability of information have been omitted for legibility reasons. In this fact table, the measurement of the current bandwidth usage is specified to be valid for at most 30 seconds, after which it is removed from the fact table.

Support for Adaptation. Storage can be unnecessary or outsourced to a device with more storage capacity. In the latter case, information requests have to be sent to a third party by using the *Context Interface* of a service. There is no requirement that all components have to execute on the same device.

4.4 Context Manipulation

This part transforms and reasons on context information to provide suitable information for initiating the context-aware service adaptation.

Context Transformation. Context transformation changes the way certain information is represented. For example, a temperature expressed in $^\circ C$ can be transformed into $^\circ F$ using simple mathematical transformation rules. Classification is another kind of transformation, where accuracy of information is given up for the sake of more meaningful information. For example, the geographical location specification in Table 1 using longitude and latitude coordinates can be replaced by the nearest major city, in this case Brussels, resulting in a better human understanding of the location. Classification is more complex compared to the mathematical equivalences as it requires extra information defining the categories and a general distance function to select the category that fits best.

Context Reasoning. Context reasoning derives new information based on existing facts and derivation rules. Whereas context transformation changes the way a concept is expressed, context reasoning combines derivation rules and facts into other facts which were only available implicitly. For example, a calendar service with information on events combined with the current time allows to predict the current location of a person. Initially, we investigated Racer and Jess as reasoning tools, but due to technical integration issues we chose for the Jena 2 Ontology and Reasoning subsystem [18] for context manipulation.

Context-Based Decision Making and Adaptation. The decision making on service adaptation may require manipulation of context information. The adaptation is initiated by several triggers that fire due to a change in context. The necessary information is delivered by the context repository, or it is deduced after several transformation and reasoning steps.

Component-Based Context Manipulation. A general overview of the context manipulation is given in Figure 5. The *Context Transformation* component and the *Context Reasoning* component are able to derive new facts by combining information from the *Fact Container* and the *Ontology Container*. These new facts are stored again in the *Fact Container*. The *Resource Monitor* is responsible for enabling garbage collection on the *Fact Container* when running low on storage capacity. The *Context Dissemination* component is responsible for providing the necessary information to the triggers (not shown in the figure) that activate service adaptation. If certain necessary context information is not present or cannot be derived, then the *Context Dissemination* component is also able to send a request to third parties using the *Context Interface*. For example, by providing name and address information as possible inputs, the *Context Dissemination* component can send an information request to a *Yellow Pages* service that is able to provide mobile phone contact information.

Support for Adaptation. The elimination of unnecessary transformation and reasoning components results in increased storage space as the transformation rules are no longer required. If, however, context transformation or reasoning is

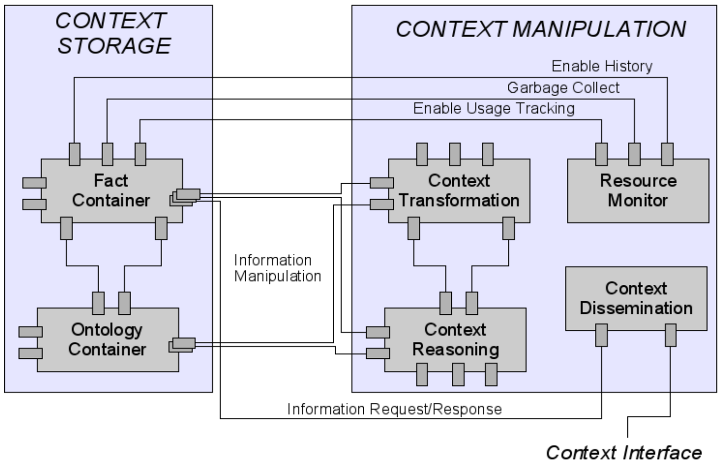


Fig. 5. Overview of the *Context Manipulation* component

required but is too resource intensive on the current device, then the necessary facts, ontologies and reasoning can be delegated to a more powerful device.

5 Evaluation, Current Status and Future Work

The component-based management system has been implemented to a large extent (user profiling is not yet supported) on top of Draco [19], an extensible runtime system for components in Java designed to be run on advanced embedded devices. The base system supports extensions for component distribution [11], live updates [9], contract monitoring and resource management [10] and provides a unique test platform for validating the proposed concepts in a pervasive and ubiquitous computing context.

Several information providers, the storage components, and the transformation components are operational. A test case showed that the advantage of regaining storage space by eliminating old information is minimal, as the resource requirements for the libraries responsible for reasoning on OWL ontologies are much higher than the storage capacity needed for small scale services (> factor 100). A small overview of the memory usage for the deployment of several components just before their activation is given in Table 2. It demonstrates how much memory can be saved when certain components are eliminated.

Table 2. Memory requirements for deploying a specific component

Component	Type	Memory
`Weather`	Sensor	6264 bytes
`Clock`	Sensor	7024 bytes
`Relevance Filter`	Retrieval	11232 bytes
`Fact Container`	Storage	23484 bytes
`Context Transformation`	Manipulation	123766 bytes
`Context Reasoning`	Manipulation	1080944 bytes

Future work will focus on the modeling of resource requirements for context transformations and context derivations. This is useful if a user has a preference for receiving a rough but quick response, or for receiving a more accurate answer for which he is willing to wait a bit longer. This work will result in a better validation of the component-based context management system, because in the current system it is difficult to define an optimal deployment without having even a rough estimate of the processing time for all context management activities.

6 Related Work

Research on context-awareness has already been focusing on service adaptations in the past. The CoBrA architecture [20] is a context broker that uses ontologies and maintains a shared model of context on behalf of a community of agents, services, and devices in a certain environment. It also provides privacy protection for the users in the space by enforcing the policy rules that these users define. The difference between this and our system is that CoBrA manages the context for all computing entities in a certain environment, instead of each device managing its own context. The advantage of our approach is that it provides better support for mobility in ubiquitous and pervasive computing environments.

Efstratiou et al. [21] also propose a service platform with support for context-aware service adaptation. The authors describe the architectural requirements for adaptation control and coordination for mobile applications. In our paper, we have not only shown how services can be adapted, but also how the driving force behind adaptation, i.e. the context management system, can be adapted to different working conditions, including support for distributed execution.

The M3 architecture [22] is an open component-based architecture for pervasive systems that supports context management and adaptation, and includes a coordination language to coordinate system events. The context manager uses RDF to describe devices and user context. The context manager of M3 does not support context-based self-adaptation.

7 Conclusions

This paper presents a component-based approach for managing context information. The novel contribution is that the management system itself can be adapted to a device's capabilities or service requirements by enabling or disabling certain components or specific properties of certain components.

A further advantage is that these components do not necessarily have to execute on the same device. In the event of excessive storage or processing power demands, the context management system can be distributed, if necessary by relocating components to other devices.

Future work will focus on the modeling of resource requirements for the transformation and reasoning components, so that processing time can be estimated and an optimal deployment of context operations can be achieved to further increase the quality of service of our context management system.

References

1. Dey, A.K., Abowd, G.D.: Towards a better understanding of context and context-awareness. In: Workshop on The What, Who, Where, When, and How of Context-Awareness, Conference on Human Factors in Computer Systems (CHI2000). (2001)
2. Satyanarayanan, M.: Pervasive computing: Vision and challenges. IEEE Personal Communications (2001) 10–17
3. Dey, A.K., Salber, D., Abowd, G.D.: A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. Human-Computer Interaction (HCI) Journal **16** (2001) 97–166
4. Strang. T., et al.: CoOL: A Context Ontology Language to enable Contextual Interoperability. In Stefani, J.B., Dameure, I., Hagimont, D., eds.: LNCS 2893: Proceedings of 4th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS2003). Volume 2893 of Lecture Notes in Computer Science (LNCS)., Paris/France, Springer Verlag (2003) 236–247

5. Gu, T., et al.: An ontology-based context model in intelligent environments. In Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference, San Diego, California, USA (2004)
6. Chen, H., Finin, T., Joshi, A.: An ontology for context-aware pervasive computing environments. Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review (2003)
7. Preuveneers, D., Van den Bergh, J., Wagelaar, D., Georges, A., Rigole, P., Clerckx, T., Berbers, Y., Coninx, K., Jonckers, V., De Bosschere, K.: Towards an extensible context ontology for Ambient Intelligence. In: Proceedings of the Second European Symposium on Ambient Intelligence, Springer (2004)
8. Urting, D., Van Baelen, S., Holvoet, T., Berbers, Y.: Embedded software development: Components and contracts. In: Proceedings of the IASTED International Conference Parallel and Distributed Computing and Systems. (2001) 685–690
9. Vandewoude, Y., Berbers, Y.: Run-time evolution for embedded component-oriented systems. In Werner, B., ed.: Proceedings of the International Conference on Software Maintenance, Canada, IEEE Computer Society (2002) 242–245
10. Wils, A., Gorinsek, J., Van Baelen, S., Berbers, Y., De Vlaminck, K.: Flexible Component Contracts for Local Resource Awareness. In Bryce, C., Czajkowski, G., eds.: ECOOP 2003 Workshop on resource aware computing. (2003)
11. Rigole, P., Berbers, Y., Holvoet, T.: Mobile adaptive tasks guided by resource contracts. In: the 2nd Workshop on Middleware for Pervasive and Ad-Hoc Computing, Toronto, Ontario, Canada (2004) 117–120
12. Wagelaar, D.: Context-driven model refinement. In: Proceedings of the MDAFA 2004 workshop, Linköping, Sweden (2004)
13. Korpipää, P., et al.: Managing context information in mobile devices. IEEE Pervasive Computing, Mobile and Ubiquitous Systems **2** (2003) 42–51
14. Indulska, J., et al.: Experiences in using cc/pp in context-aware systems. In Stefani, J.B., Dameure, I., Hagimont, D., eds.: LNCS 2893: Proceedings of 4th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS2003). Volume 2893 of Lecture Notes in Computer Science (LNCS)., Paris/France, Springer Verlag (2003) 224–235
15. Buchholz, S., Hamann, T., Hubsch, G.: Comprehensive structured context profiles (cscp): Design and experiences. In: Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops. (2004)
16. DistriNet (K.U.Leuven), EDM (LUC), ELIS-PARIS (UGent), PROG (VUB) and SSEL (VUB): CoDAMoS: Context Driven Adaptation of Mobile Services. (`http://www.cs.kuleuven.ac.be/cwis/research/distrinet/projects/CoDAMoS/`)
17. Senart, A., Bouroche, M., Biegel, G., Cahill, V.: Component-based middleware architecture for sentient computing. In: Workshop on Component-oriented approaches to Context-aware computing, ECOOP '04, Oslo, Norway (2004)
18. HP Labs: Jena 2 - A Semantic Web Framework. `http://www.hpl.hp.com/semweb/jena2.htm` (2004)
19. Vandewoude, Y., Rigole, P., Urting, D., Berbers, Y.: Draco : An adaptive runtime environment for components. Technical Report CW372, Department of Computer Science, Katholieke Universiteit Leuven, Belgium (2003)
20. Chen, H.: An intelligent broker architecture for context-aware systems. http://cobra.umbc.edu/ (2003)

21. Efstratiou, C., Cheverst, K., Davies, N., Friday, A.: An architecture for the effective support of adaptive context-aware applications. In: Proceedings of 2nd International Conference in Mobile Data Management (MDM‘01). Volume Lecture Notes in Computer Science Volume 1987., Hong Kong, Springer (2001) 15–26
22. Indulska, J., Loke, S., Rakotonirainy, A., Witana, V., Zaslavsky, A.: An open architecture for pervasive systems. In: Proceedings of the Third IFIP TC6/WG6.1 International Working Conference on Distributed Applications and Interoperable Systems, Kluwer (2001) 175–187

Customised Billing for Location-Based Services

Maria Koutsopoulou[1], Spyridon Panagiotakis[1], Athanassia Alonistioti[1], and Alexandros Kaloxylos[2]

[1] Communication Networks Laboratory,
Department of Informatics and Telecommunications,
University of Athens,
Panepistimioupolis, 157 84 Athens, Greece
{mkoutsop, spanag, nancy}@di.uoa.gr
[2] Department of Telecommunications Science and Technology,
University of Peloponnesus
22 100 Tripoli, Greece
kaloxyl@uop.gr
http://cnl.di.uoa.gr/cnluk/cnlindex.php

Abstract. As the new communications era aims to provide mobile users with advanced and customised services, it becomes apparent that the requirement for intelligent charging and customised billing of service access emerges bravely. In that scope, the enrichment of mobile services with attributes related to user profiles, context, location, presence and other elements providing customization, affects also the charging process. In particular, the charging and billing for service access have to be more flexible and personalised, so that especial, customised billing for new location-based services and applications is enabled. Hence, charging information related to the user profile and preferences as well as the user location will also have to be encountered at the billing process. This paper presents an integrated architecture, which allows flexible charging and customised billing for the usage of location-based services.

1 Introduction

The emergency services offered by mobile network operators since 1998, was the first kind of location-based services. The cell-wide positioning information provided by that networks was not accurate, since the size of cells varies notably depending on location area due to differences in terrain and population density. Nowadays that the mature of positioning technologies enables the detection of persons and objects with much greater accuracy, mobile users start carrying their terminal devices (i.e. cellular phones, PDAs and laptops) not only for emergencies but basically for work and entertainment purposes. Hence, the user location detection is mainly used anymore for the provision of location-based information, guidance, and entertainment services.

Unlike applications that constitute transformation of applications originally developed for personal computers accessing the World Wide Web, to the constraints of mobile devices, location-based services take full advantage of the mobile device's portability. In that sense, location-based services may change the way consumers come to depend on their mobile equipment, since mobile users can get highly location-specific, real-time information at the precise moment they are making a decision. For example, a driver can be alerted to alternative traffic routes on a turn-by-turn basis, a traveler can be informed of the nearest hotel at night in an unknown city, etc. In practice, mobile network operators, service and content providers, equipment vendors, handset manufacturers and software developers need to co-operate to bring efficient location-based services in the daily life of mobile consumers. The provision of advanced location-based services is a big opportunity for all business players involved to increase their revenues. Therefore, charging and billing for location-based services are crucial issues that should be addressed.

The charging criteria for location-based services seem to be different from the ones traditionally used in mobile networks such as measuring the connection time or data volume. Definitely, the time- or volume-based charging can of course still be used for the transport part of provided location-based services. However, the charging information related to the usage of location-based services concerns mainly the added value of such a service and content and is additional information to the transport level information. The introduction of location-based services in service provisioning opens up a large variety of new pricing models that can apply. It should be mentioned here, that apart mobile users the location-based service providers could also constitute potential charged parties. For example, push initiators can be charged for the services they offer.

This paper addresses the charging issues related to location-based services and introduces an integrated architecture that allows flexible charging and customised billing for advanced location-based services usage. In particular, next section introduces location-based services, and Section 3 discusses some charging issues related to location-based services. Section 4 describes the proposed integrated charging architecture, while an indicative example of its functionality is given in Section 5 and finally, Section 6 concludes the paper.

2 Typical Categories of Location-Based Services

The term “Location-based services” refers to services offered to mobile users in which the user location information is used in order to add value to the service as a whole. Currently the most common location-based services include “Yellow Pages” and “Point of Interest” that allow users to find the nearest hotel, pharmacy or cinema based on their current location. Other common location-based services provide navigation and traffic conditions information. Typical location-based services can be distinguished in five categories [1]:

- <u>Traffic coordination and Management</u>
  Such services identify traffic jams and taking into account the current location of subscribers inform the subscribers driving around. Alternatively, they can determine the fastest route between two positions at the moment, estimate the total travel time, or provide alternative routes for the remaining travel.

- Location-aware advertising and general content delivery
  Whenever a subscriber indicates to the service that he is in "shopping-mode", he receives sales information (e.g. discounts) based on his current location. Location information is used along with his shopping preferences included in an associated shopping profile so that better advertising service is provided (e.g., the user receives only advertisements related to his preferences).
- Integrated tourist services
  Such services may include from advertisement of the available options for numerous tourist services and guided tours, to accommodation information, transportation, cultural events and museum guides.
- Safety-related services
  It is possible to monitor tourists or workers traveling in dangerous terrain and guide them to desired destinations along safe paths.
- Location-based games and entertainment
  Examples of such services are the treasure hunting, and the location-based ICQ service.

3 Charging Issues Related to Location-Based Services

Much effort have been put by the 3G Partnership Program (3GPP) [2] and the Open Mobile Alliance (OMA) [3] in standardizing location-based services. The 3GPP mainly focuses on the development and promotion of common and ubiquitous solutions for Location Services, which should be network independent. In particular, 3GPP has specified the stage 2 of the LoCation Services (LCS) feature in UMTS and GSM [4], which provides the mechanisms to support mobile location services for operators, subscribers and third party service providers. This service description [4] covers the LCS system functional model for the whole system, the LCS system architecture, state descriptions, message flows, etc., along with the operations performed by the LCS Server. The LCS Server is considered to be a software and/or hardware entity offering LCS capabilities. It accepts services requests and sends back responses to the received requests. The LCS server consists of LCS components (e.g., the GMLC), which are distributed to one or more PLMN and/or service providers [4].

On the other hand, the OMA continues the technical activities originated in the Location Interoperability Forum (LIF) [5] for the specification of the Mobile Location Protocol (MLP) [6]. The MLP is an application-level protocol for obtaining the position of mobile stations (mobile phones, wireless personal digital assistants, etc.) independent of underlying network technology. The MLP serves as the interface between a Location Server and a location-based service/application, acting as LCS client. Possible realization of such a Location Server is the 3GPP GMLC [4], which act as the front-end of the LCS Server defined in GSM and UMTS by 3GPP. In the most scenarios an LCS client initiates the dialogue by sending a query to the Location Server and the server responds to the query. The OMA MLP specification [6] defines the core set of operations that a Location Server should be able to perform.

Till now the LCS Server incorporated in the mobile operator's infrastructure provides charging information related to the LCS features usage. The charging information collected by the mobile operator is used only to charge LCS Clients for receiving information about the user location. In addition, 3GPP [7, 8] and researchers [9] have thoroughly studied an advanced location-based service providing subscribers with customised billing, the location-based charging. More specifically, the location-based charging is a flexible model that takes into account location information provided by the LCS Server of the network operator to provide subscribers with a customised charging scheme depending on its location or geographic zone. To elaborate, specifically, this service could apply reduced rates to those areas most often frequented by the subscriber by taking into consideration the subscriber's daily route and life style. For example, a “home” zone may be defined around a user’s home, work or travel corridor. Additionally, different rates may be applied in different zones based on the time of day or week. Location-based Charging should analyze location information to compare against “home” zones established for the subscriber. The service would notify the subscriber of its relative location to the established “home” zones, indicating either "in" or "out" of zone, along with the associated tariffing and pricing policies.

The existing approaches in charging and location-based services attempt to cover different needs. Definitely, the “charging of location-based services” is something different from the “location-based charging” and the charging of the LCS Client for receiving information on the location of users. In order to enable customized charging and billing for location-based services, some unresolved charging issues remain.

Firstly, additional charging information in forms of Charging Data Record (CDR) is required [7]. The following additional information should be included:

- Type of the location-based service
- Identity of the location-based service
- Required location information accuracy
- Time stamps
- Type of user equipment

Additionally, the CDR should indicate the charged party, i.e. normally the calling party. Alternatively, it is possible the application of reverse charging or the charged party not to be involved in the chargeable event (e.g. a company). It should be possible for multiple leg calls (e.g. forwarded, conference or roamed) each party to be charged as if each leg was separately initiated. In any case the charged party should be provided with charging information accurately, and in time, so that is informed about expected call charges. Finally, the location-based service usage should be recorded on event, call or session basis [10]. This brings additional requirements for the management of the charging process.

Hence, the need for a solution that will manage all aspects related to the charging process for location-based services emerges. In that context, we propose the introduction of an integrated charging architecture that caters for all players involved in location-based services provision, manages all aspects related to the measurement and recording of location-based service usage, and enables customised billing for such services. Next section elaborates on the proposed architecture.

4 Integrated Charging Architecture

The proposed charging architecture, which is already prototyped in SDL (Specification and description Language) and illustrated in Fig. 1, is ambitiously regarded that will cover the aforementioned unresolved charging issues.

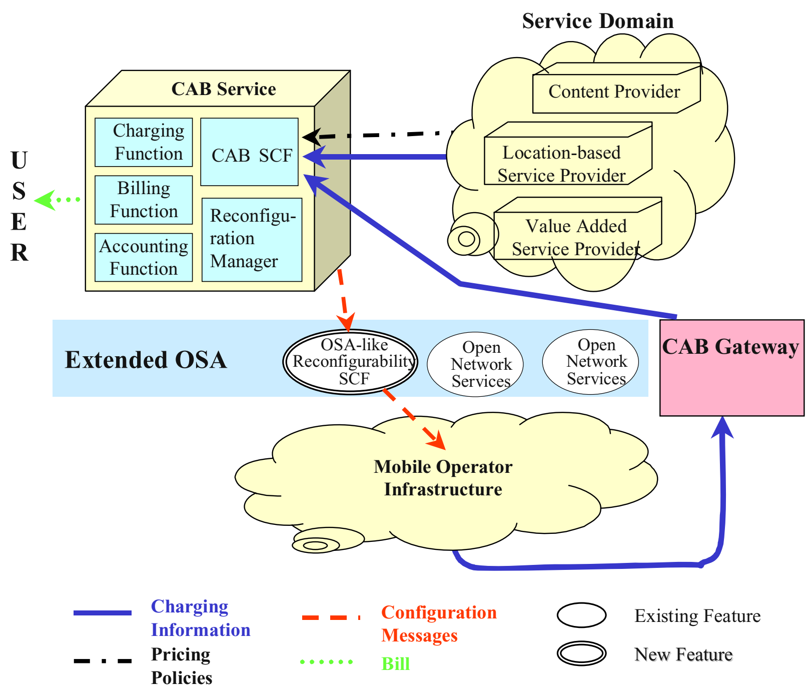


Fig. 1. Integrated charging architecture for customised billing

Definitely, the provision of location-based services enriches the traditional business players comprising the mobile service provisioning chain (i.e. network operators, service providers, content providers) with new ones (i.e. location-based service providers, location information brokers, location information interpreters, etc.). All players, old and new, should be able to participate in the control and cost sharing of the provided services. To bypass a complicated charging architecture, a layer-based charging approach can be adopted. According to this approach the charging architecture should be structured in three layers: transport, service and content. The management and processing of the relevant information should be made separately for each layer. Furthermore, different charging models should be possible to apply on each charging layer. The additional information required for customised billing of location-based services can be provided either by the mobile operator or by the respective provider. Finally, the location-based service provider is responsible to define the charged party and the applied pricing policy for service usage.

The introduced architecture supports the layer-based charging approach and enables all business players involved to submit their charging records on-line, to define the pricing policies dynamically, and to apportion their revenues automatically.

The charging information produced by the mobile network operator components and the third party service provider infrastructure is collected by a discrete service providing advanced charging mechanisms, flexible pricing and customised billing. The Charging, Accounting and Billing (CAB) service can be either under the administrative domain of one of the involved parties (i.e. mobile operator, value added service provider, location-based service provider, etc.), or it belongs to an independent third trusted party (e.g. charging/payment provider) that has the responsibility and authorization for the overall charging procedure.

The use of open APIs among players belonging to different administrative domains is regarded as the necessary mean that will enable the configuration of network entities for the collection of all required information. For example, the standardized OSA interfaces [11] enabling independent players to retrieve the user location information from the underlying mobile operator infrastructure via the respective OSA SCF [12] could be used. Furthermore, the introduction and provision of a set of open APIs for the support and management of charging related reconfiguration actions (e.g., for pricing policies updates) and the deployment of advanced charging services is essential.

The CAB service depicted in Fig. 2 comprises the following functional entities:

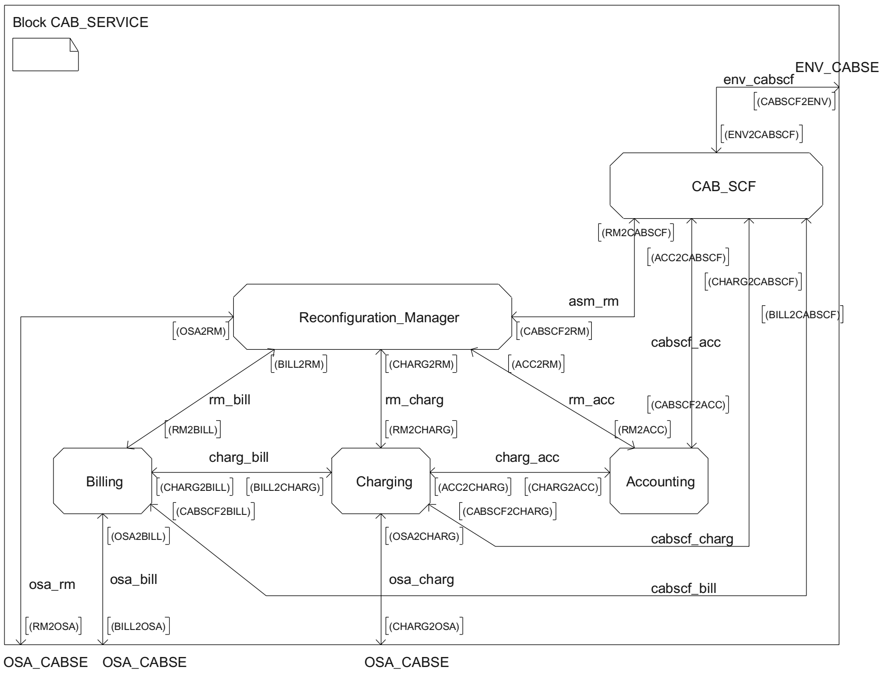


Fig. 2. Internal modules of the CAB service

- The CAB Service Capability Feature (SCF) is the functionality offered by the CAB service that is accessible via the proposed open APIs. It enables authorized business players to register new services to the CAB service, to refine the parameters of a registered service, to define pricing policies dynamically, and to submit charging records. Additionally, it provides authorized entities with advanced charging services such as location-based charging, on-line charging indication, current balance of user bill and on-line provision of statistical information. The CAB SCF incorporates the well-defined Charging SCF for content charging [13].
- The Charging module receives and processes charging information from the network elements via the CABG and the usage data and content charges from authorized independent providers via the CAB SCF. Based on the applied layer-based model, the charging information and usage data are correlated and processed. After this step transport, service, and content records are generated.
- The Billing module applies the pricing policies to the transport and service records in order to calculate the charges. The applied pricing policy depends on user, service or session characteristics and can be different from layer to layer. In addition, the billing process includes the content charges and produces a bill requiring payment.
- The Accounting module is an automatic procedure for sharing of charges and revenues between involved business entities.
- The Reconfiguration Manager module includes intelligent mechanisms for identifying the particular high-level requirements of the business players and mapping them to appropriate reconfiguration actions on the underlying network infrastructure. Specifically, in case of service registration it configures the underlying network elements to monitor the related IP flows and produce usage records according to the applied metering policy. Then, based on the specific pricing policy it configures the billing and accounting modules accordingly. Furthermore, it supports the dynamic modification of the services parameters and the applied pricing policies reconfiguring the aforementioned modules appropriately. Finally, it configures the charging module to generate charging sensitive alerts when the charging information meets some conditions, so that it provides the authorized and subscribed entities with specific event notifications (e.g. modification of tariffs).

In order to have a single logical interface between the network elements related to charging and the CAB service for the charging records transmission, we introduce a charging accounting and billing gateway (CABG). This executes a first correlation of the collected chargeable events and transfers them to the CAB service. The entities that collect and process the charging information concerning the usage of network resources (i.e. CGF and AAA) and the services' consumption (i.e. CCF and MDs) support different protocols and interfaces. The CABG receives the charging records using the respective protocols over the existing interfaces, correlates the records related to a specific chargeable event and transmits them using an open standard API to the CAB service.

In the case of roaming users, the accounting module is responsible for apportioning charges between the home environment, the serving network, and the user, and then calculating the portion that is due to each operator. The transport records concerning a roaming user are forwarded to its home network operator using the transferred account procedure (TAP) and a specific TAP format. The transfer of TAP records between the visited and the home mobile networks may be performed directly, or via a clearinghouse. Clearing-houses are independent business players responsible for TAP records creation, tariffing, and re-tariffing.

5 Customised Billing for Location-Based Services

In this section we present a part of the functionality of the proposed architecture with the execution of an indicative example scenario. This example, deals with the charging, billing and accounting process during the execution of a location-based service offered by an independent VAS provider. Specifically, the user accesses a service, which identifies traffic jams and taking into account the user's current location provides him with alternative routes. For such services the user has to pay for the transport, service and content part, while the VAS provider is charged for the LCS features usage. Furthermore, the user requests to be informed about the charging status of the executing location-based service. Moreover, the VAS provider submits to the CAB his own charging information for the provided contents with added value.

Fig. 3 presents the Message Sequence Chart (MSC) for this scenario. For the shake of simplicity, the MSC does not contain the messages exchanged between the user, the VAS provider and the Location Broker, the messages exchanged between the underlying network components and the CABG, as well as the acknowledgement messages.

More specifically, at first each authorized entity should create a session with the CAB service in order to use the services offered by the CAB SCF (CREATE_CAB_SERVICE_SESSION). Following the successful creation of the session, the user requests to be informed the applicable charges (ON_LINE_CHARGING_ INDICATION) providing the user identity and the charge unit that defines when a notification should be sent to the user.

The reconfiguration manager receives the request (through the CAB SCF) and configures the billing module to notify it when a certain limit is reached (CHARGE_EVENT_NOTIFICATION). During the execution of the service the CABG sends the charging information (CHARGING_ RECORD), collected by the underlying network elements, to the charging module. The charging module correlates the collected charging information and generates a TRANSPORT_RECORD and a SERVICE_ RECORD that are sent to the billing module.

In parallel, the VAS provider retrieves the user's location from the Location Broker; this has as result the Location Broker to send its service charges (CHARGING RECORD) to the charging module (through the CAB SCF). This information is processed and transmitted to the billing module (SERVICE_RECORD) in order the VAS provider to be charged for the LCS features usage and the accounting module

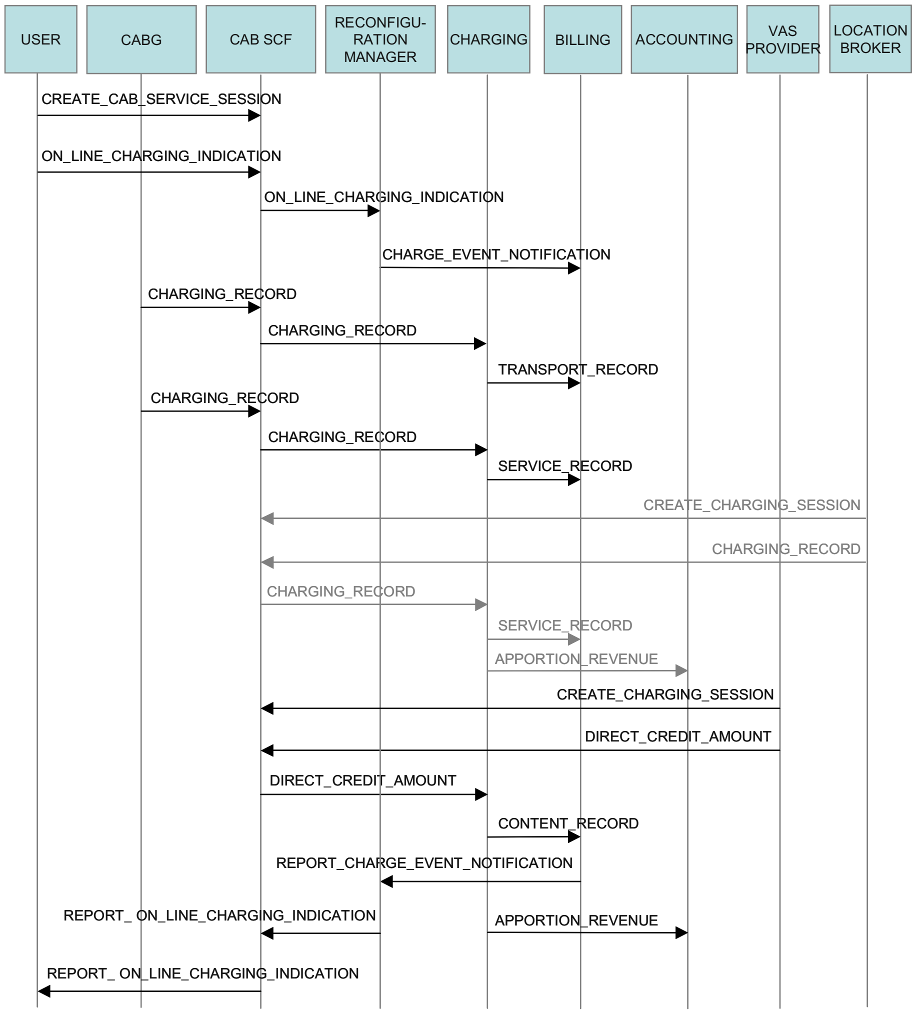


Fig. 3. CAB's functionality example

(APPORTION_ REVENUE) in order to calculate the revenues of the Location Broker (these interactions are presented in grey in Fig. 3).

Additionally, the VAS provider sends its content charges (DIRECT_CREDIT_AMOUNT), through the OSA/Parlay charging interface provided by the CAB service, to the charging module (through the CAB SCF) as well. The charging module correlates this information with the one sent by the operator and notifies appropriately the billing module (CONTENT_RECORD).

The billing module processes then the received information applying the appropriate pricing models and calculates the transport, service and content charges that concern the location-based service usage. In case that a certain limit (the charge unit) is overcame, the billing module informs the reconfiguration manager about (REPORT_CHARGE_EVENT_ NOTIFICATION) and following the requesting user is notified (REPORT_ON_LINE_CHARGING_ INDICATION). In parallel, the charging module provides the accounting system with the required information (transport, service and content charges) to apportion the revenues between the players (APPORTION_ REVENUE).

6 Conclusion

By summarizing, this paper addresses the main unresolved charging issues for location-based services and introduces an integrated architecture, which allows flexible charging and customised billing for advanced location-based services usage. The proposed CAB service supports one-stop billing schemes for the end users as well as the separation of charging events based on content, service, and transport usage information. Moreover, it enables the automatic apportioning of incomes among the players. Finally, the CAB service allows flexible charging and customised billing for advanced location-based services usage.

Acknowledgment

Work presented in this paper has been performed in the framework of the project LIAISON (IST-511766, web site http://liaison.newapplication.it/), which is partly funded by the European Community. The Authors would like to acknowledge the contributions of their colleagues. The content of this paper expresses solely the opinion of the authors.

References

1. C. S. Jensen, A. Friis-Christensen, T. B. Pedersen, D. Pfoser, S. Saltenis, and N. Tryfona "Location-based services -A database perspective" In Proceedings of the 8th Scandinavian Research Conference on Geographical Information Science (ScanGIS2001), 2001
2. Third Generation Partnership Project (3GPP). http://www.3gpp.org
3. Open Mobile Alliance (OMA). http://www.openmobilealliance.org
4. 3rd Generation Partnership Project (3GPP) TS 23.271: "Functional stage 2 description of LCS", version 6.7.0, 2004-09
5. Location Interoperability Forum (LIF), http://www.locationforum.org/
6. Open Mobile Alliance (OMA) OMA-LIF-MLP-V3_1-20040316-C: "Mobile Location Protocol (MLP)", Candidate Version 3.1, 16 Mar 2004
7. 3GPP TS 22.071, 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Location Services (LCS); Service description; Stage 1
8. 3GPP TS 22.115: 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Service aspects; Charging and Billing
9. S. Panagiotakis, M. Koutsopoulou, A. Alonistioti, A. Kaloxylos, "Generic Framework for the Provision of Efficient Location-based Charging over Future Mobile Communication Networks", PIMRC, Lisbon, Portugal, September 2002

10. Open Mobile Alliance, "WAP Billing Framework", OMA-WBF-v1_0, http://www.openmobilealliance.org/
11. 3GPP TS 29.198-1, 3rd Generation Partnership Project; Technical Specification Group Core Network; Open Service Access (OSA); Application Programming Interface (API); Part 1: Overview
12. 3GPP TS 29.198-6 version 5.4.0, 3rd Generation Partnership Project; Technical Specification Group Core Network; Open Service Access (OSA); Application Programming Interface (API); Part 6: Mobility
13. 3GPP TS 29.198-12 version 5.4.0, 3rd Generation Partnership Project; Technical Specification Group Core Network; Open Service Access (OSA); Application Programming Interface (API); Part 12: Charging

Loosely-Coupled Integration of CSCW Systems

Roberta L. Gomes[*], Guillermo J. Hoyos Rivera[**], and Jean Pierre Courtiat

LAAS-CNRS,
7, Avenue du Colonel Roche, 31077 Toulouse Cedex 4, France
{rgomes, ghoyos, courtiat}@laas.fr

Abstract. As collaboration activities usually involve several people with different group tasks and needs, they are rarely supported by a single CSCW technology. Then different types of collaborative applications are usually applied in order to support group work. But in spite of being used to accomplish a common collaboration task, these applications are executed independently without getting any advantage of each other. The integration of such applications would allow them to dynamically interoperate, combining their different functionalities in a controlled way. In order to achieve integration, we propose LEICA[1], a loosely-coupled integration environment which allows collaborative applications to interact without loosing their autonomy. LEICA relies on Web services technology, event notification system, and collaboration policies for controlling the interactions between integrated applications.

1 Introduction

The increasing communication demands of geographically dispersed organizations combined with the technological advances in networking have given rise to the development of a great number of CSCW (Computer Supported Cooperative Work) systems. These systems aim to provide suitable forms of cooperation within a group of users to perform collaborative activities.

As collaboration activities usually involve several people, with different group tasks and needs, they are rarely supported by a single technology. Thus some CSCW systems try to combine different functionalities in order to support collaboration into a single environment. The main weakness of this approach is the challenge behind the anticipation of all the requirements of cooperative situations. This way a CSCW system is hardly suitable and sufficient for every collaborative activity.

As a result, current collaborative environments consist of a range of applications, working side by side but independently, without really getting advantage of each other. Allowing the integration of these applications could bring significant benefits to users. An integrated collaboration environment would allow the different functionalities of existing applications to be dynamically combined and controlled (enhancing

[*] Researcher supported by a scholarship from CNPq, Brazil.
[**] Researcher supported by scholarship 70360 CONACyT/PROMEP, Mexico.
[1] This research is partially supported by the European IST project *Lab@Future*.

flexibility and tailoring possibilities).

In order to achieve the integration of existing CSCW systems avoiding dealing with their low-level features, we propose LEICA, a "Loosely-coupled Environment for Integrating Collaborative Applications". Relying on Web services technology [1] and an event notification system (supporting the publish/subscribe paradigm [2]) different collaborative applications can interoperate by exchanging information within the context of a global *SuperSession*. The loosely-coupled approach proposed by LEICA overcomes two problems usually related to integration environments: (i) it does not require a true semantic integration of collaborative applications, (ii) once integrated to the environment, collaborative applications keep their autonomy.

The definition of collaboration policies controls the interactions among integrated applications, *i.e.* how the collaboration activity supported by one application will be affected by information received from other applications. In practice, these applications interact through the notification of events which may lead to performing specific action(s) in some of these applications.

The interaction degree among integrated applications depends obviously on the nature of the events they are able to exchange, and actions they are able to perform. Three main cases may be considered when integrating applications: a) open source applications, b) API-based applications, and c) applications without any API. Integration of open source applications can achieve the tightest interaction degree, since any internal event/action can be exported/performed; it might however imply great development efforts. Integration of API-based applications is straightforward, and interaction is limited to the provided API. Applications providing no API are constrained to interact only through application *start* and *stop* actions. LEICA's integration approach has been mainly driven by the second case (b). We believe that developers are certainly interested in creating specific and performable collaboration tools that can be used either stand-alone or integrated with other applications (through a flexible API), thus being able to get a great share of the market. This is for instance the case of Skype™ [3], a successful example of communication tool that has recently released its API.

The paper is structured as follows. Section 2 presents related work regarding the integration of existing CSCW systems. Section 3 overviews the general integration approach proposed by LEICA. Section 4 describes how a *SuperSession* is configured and explains how to specify collaboration policies. Section 5 presents the LEICA's architecture and how to integrate applications in practice. Section 6 draws some conclusions and presents directions of future work.

2 Related Work

In [4], Dewan addresses basic issues in interoperating heterogeneous CSCW systems that concurrently manipulate the same artifacts. However, it does not regard the interoperation of CSCW systems that, despite being involved in the same collaborative tasks, do not deal with the same artifacts (*e.g.* videoconference and shared whiteboard).

In [5] authors propose an integrative framework based on a three-level model: ontological, coordination and user interface. An internal knowledge of the collaborative application is needed so that its functionalities can be mapped into the three-level model in order to achieve integration. Accordingly, the integration of third party applications becomes a complex (even impossible) task.

In [6] authors present the CVW, a prototype collaborative computing environment defining a place-based system for integrating document and meeting-centric tools. Basic freeware collaborative applications have already been integrated, and new, special-purpose tools can be integrated, but the integration process is not straightforward – tools must be designed against a place-based API.

Systems like AREA [7] and NESSIE [8] have tried to propose a loosely-coupled integration for supporting cross-application awareness. Like LEICA, these systems are based on the exchange of activity relevant events. However these environments just aim to provide users with a common awareness of the whole collaboration activity. They do not provide any means for defining how an application should react when events are notified by other applications.

Another proposal also based on a loosely-coupled approach is presented in [9]. The authors define a framework where Web services are used to wrap collaborative applications in order to integrate them. Since they leverage open Internet standards, Web services overcome the interoperability issues usually associated with more general integration solutions like CORBA [10], DCOM [11] and EJB [12]. Besides, they are simpler to design, develop, maintain and use. Web services based integration is quite flexible, as it is built on a loose coupling between applications.

One of the drawbacks related to the Web services wrapping approach (in particular to the use of SOAP [13]) is that it represents an additional tier causing some overhead in processing exchanged messages [14,15]. Besides, depending on the architecture of the existing collaborative applications, the complete wrapping of these applications as Web services may imply great development efforts or even applications redesign.

Therefore, unlike the approach employed in [9], and following the recommendations of [14] and [15], we decided to use Web services for coarse-grained operations only. Thus, LEICA applies Web services as an initial mechanism for (i) registering newly integrated applications, (ii) setting and (iii) starting up collaborative sessions. Then, a different infrastructure is used to implement the event notification system in charge of interconnecting the collaborative applications during the execution of an integrated collaborative session. An overview of the proposed integration approach is presented in the following section.

3 General Integration Approach

The integration of a collaborative application to LEICA is achieved by attaching a *Wrapper* to this application[2]. This *Wrapper* comprises a Web services interface allowing the collaborative application to register itself with LEICA[3] as an integrated appli-

[2] *Wrappers* are attached to servers of client/server and multi-server applications, and to the peers of peer-to-peer (P2P) applications.

[3] Except for P2P applications, which use a *P2P Proxy* in order to register themselves.

cation. Through its Web services ports, the integrated application can interact with the *Session Configuration Service* (fig. 1).

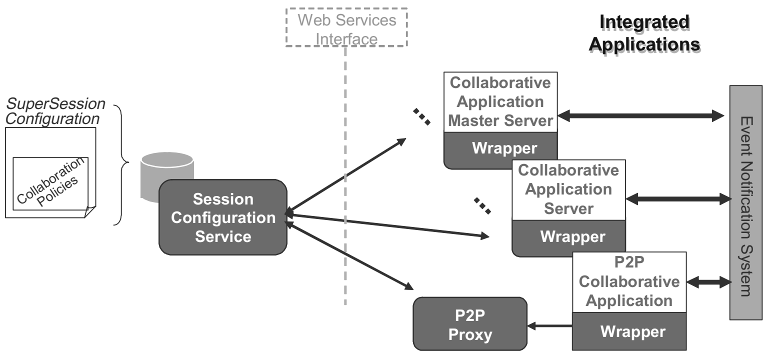

Web Services Interface
Integrated Applications
SuperSession Configuration
Collaboration Policies
Session Configuration Service
Collaborative Application Master Server
Wrapper
Collaborative Application Server
Wrapper
P2P Collaborative Application
Wrapper
P2P Proxy
Event Notification System

Fig. 1. LEICA general integration framework

The *Session Configuration Service* is a Web service used for (i) configuring new global *SuperSessions* and (ii) starting up *SuperSessions*. A *SuperSession* is an integrated collaborative session holding the whole collaboration activity. Within the context of a global *SuperSession*, different *specificSessions* can exist. A *specificSession* is then a conventional collaborative session defined within the context of one collaborative application (*e.g.* a videoconference session, a whiteboard session, *etc.*).

During the *SuperSession* configuration process, the *Session Configuration Service* dynamically contacts each integrated collaborative application in order to request: (i) which specific data is required to create *specificSessions* for this respective application (*e.g.* a videoconference tool might need an IP multicast address); and (ii) which kind of events it can notify, and action requests it can receive. This second information will be used during the collaboration policies definition process.

Collaboration policies are a set of rules following a condition/action model. These rules define how collaborative applications might react when events coming from other collaborative applications are notified. In other words, collaboration policies are the mechanism allowing to determine how an application should react when receiving information (events) notified by other applications.

Once a *SuperSession* has been configured, the *Session Configuration Service* can finally start it. To do so, firstly it contacts each integrated collaborative application requesting them to create the *specificSessions* defined in this *SuperSession*. Then, these collaborative applications are interconnected through an event notification service (as previously explained, from this point Web services are not used anymore).

As collaboration activities progress, collaborative applications exchange event notifications in a peer-to-peer fashion. Meanwhile, *Wrappers* are in charge of managing the collaboration policies. When the *Wrapper* of a collaborative application receives event notifications, it verifies if the notified events enable any policy rule concerning this collaborative application. If so, the *Wrapper* sends action requests to the respective application. Note that LEICA is not intended to support low-level physical events (e.g. mouse click/scrolling) or high frequency synchronization events (e.g. current position of moving objects). It aims to support activity relevant events that carry some semantics.

In the next section we detail the *SuperSession* configuration process, explaining how to specify the collaboration policies of a *SuperSession*.

4 *SuperSession* Configuration

In order to create a *SuperSession*, a two steps configuration process is carried out: (i) Session Management configuration and (ii) Collaboration Policies configuration.

4.1 Session Management Configuration

In the first configuration step, all data necessary to define the main elements of a *SuperSession* are provided. Two groups of information have to be specified:

- General Session Management information (*GSMinfo*). It carries management information such as scheduling, membership and general user roles.
- Integrated Applications information (*IAinfo*). It defines the list of integrated applications to be used during this *SuperSession*. For each collaborative application, a list of *specificSessions* is defined, where specific data required by this application for creating sessions is provided (*e.g.* a videoconference application will be provided with an IP multicast address).

4.2 Collaboration Policies Configuration

The second step of the *SuperSession* configuration process deals with the specification of collaboration policies. As briefly described in section 3, these policies are responsible for linking the collaboration activities supported by different *specificSessions* in the context of the global *SuperSession*. This is carried out through a set of policy rules, basically allowing the association of *n* event notifications to the execution of *m* actions (under certain conditions).

In order to specify the policy rules, a collaboration policies editor might be used. Policy rules are then created through the composition of GUI components, called *policy widgets*. Once the collaboration policies are created, the respective XML data is generated which is appended to the *SuperSession* configuration file. The rules' semantics associated with the XML syntax has been defined using the RT-Lotos formal description technique [16]. This semantics is implemented by a sub-module of the *Wrapper*.

Figure 2 illustrates the *policy widgets* used to create policy rules. These widgets can be connected through their connection points. The basic composition rules are: (i) policy rules are read from left to right; (ii) only widgets without any connection point on their left can appear on the left end of a policy rule; (iii) only widgets without any connection point on their right can appear on the right end of a policy rule.

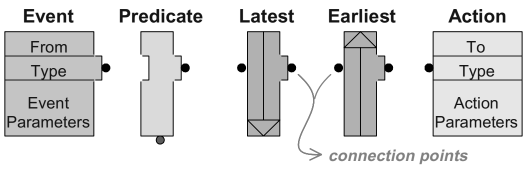


Fig. 2. *Policy widgets*

The *Event* widget represents an event notification. Each *Event* is associated with a collaborative application (field "From") and has a type (field "Type"). In the "*Event* Parameters" area it is possible to define matching patterns (filters) for parameters' values. The *Action* widget represents an action execution request. Each *Action* is associated with a collaborative application (field "To") and has a type (field "Type"). In the "Action Parameters" area, all the required parameters for this action type are specified. Note that all event and action types (and their parameters) are well-known since they are provided by each integrated application.

Figure 3 shows a simple collaboration policy rule associating directly one action with one event. This policy rule is enabled when the specified event is notified. It specifies that: if the application "CA1" notifies an *Event* of type "T1" with parameter "a:M*" (a string starting by "M"), then an action request of type "T2" must be sent to application "CA2". The '%' character is a reference operator, indicating that the *Action*'s parameters "d" and "y" have their values copied from the *Event*'s parameters "b" and "c".

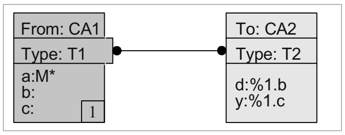


Fig. 3. A simple collaboration policy rule

A *Predicate* widget allows the association of conditions to enable policy rules. A *Predicate* can appear alone or attached to every *policy widget* but an *Action*. It contains a predicate that is specified in Java™ language syntax. *Predicates* can impose time constraints, as well as conditions based on the current *SuperSession* state. When a *Predicate* is attached to an *Event* (or to a *Latest*) it can also reference the parameters of the respective *Event* (or the parameters of the *Events* connected to the *Latest*).

The *Earliest* and *Latest* widgets allow the composition of different *Events* for the specification of a policy rule. When *Event*s are grouped through an *Earliest*, the policy rule is enabled when one of the specified *Events* is notified. When *Events* are grouped through a *Latest*, the policy rule is enabled after all events have been notified.

Figure 4 shows two examples of policy rules using *Latest* and *Earliest* widgets. In the left example, the policy rule is enabled when both specified events are notified (for the first one, the attached *Predicate* must be evaluated to true). Then the *Predicate* associated with the *Latest* widget is also evaluated. If it is true, then the two specified action requests are sent. In the right example, there is an *Event* and a *Predicate* grouped through an *Earliest*. Policy rules like this one aim at waiting the fulfillment of certain conditions to be enabled (*e.g.* regarding the *SuperSession* state); however if a particular event is notified before this condition is satisfied, then the policy rule is also enabled.

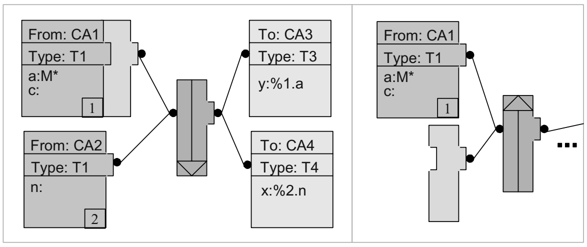


Fig. 4. Policy rules using *Latest* and *Earliest*

An upcoming problem is related to the fact of having more than one *Event* in the same policy rule. Thus, an automatic sequence number is attributed to each *Event* in order to be used as identifier while referencing event parameters. Another constraint related to event parameters referencing appears when *Events* are grouped through an *Earliest*. As it defines a non-deterministic behavior (there is no way of knowing which of the *Events* will enable the policy rule) parameters from *Events* of different types grouped through an *Earliest* can not be referenced by a *Predicate* attached to this *Earliest*.

As illustrated in figure 5, different *Earliest* and *Latest* widgets can be combined in order to create compound rules.

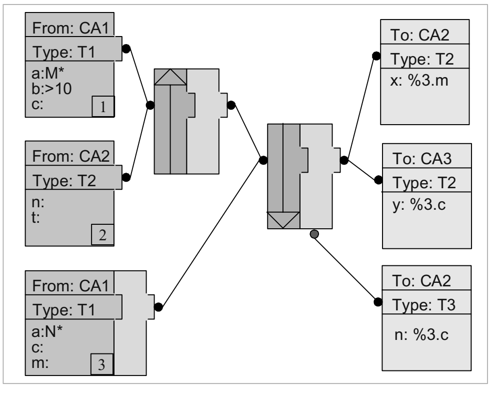


Fig. 5. A compound collaboration policy rule

5 LEICA's Architecture

Following the integration framework presented in section 3, we describe here the LEICA's architecture and different implementation aspects of a prototype currently under development. Java™ has been chosen as underlying technology for implementation. To precisely describe each architecture component, let us consider the five necessary steps to achieve the execution of a *SuperSession*.

5.1 Integrating a Collaborative Application

CSCW systems may present different distribution architectures, varying from centralized to replicated architectures. Centralized architectures are usually implemented according to the client/server or multi-server approaches. Replicated architectures are mainly implemented following the P2P approach. These three different approaches are considered to determine how to integrate collaborative applications to LEICA.

When integrating client/server or multi-server collaborative applications, a *Server Wrapper* must be added to the servers. In the case of a P2P collaborative application, a *P2P Wrapper* is used. As shown in figure 6, the difference between these two *Wrappers* deals with the Web services interface, not present in the second case. Since P2P applications are usually dynamically executed in the users' hosts when they get connected, they cannot be permanently available as Web services. To overcome this problem, a *P2P Proxy* is used.

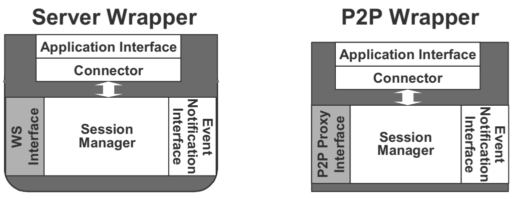


Fig. 6. The LEICA *Wrapper*s

The *Wrapper* is a Java component to be tied to the collaborative application through an *Application Interface*. This last is an abstract class to be extended in order to implement the communication interface with the collaborative application itself [4]. Through this interface the collaborative application notifies the *Wrapper* of "what is happening" inside its collaboration context (*i.e.* make event notifications), and receives all s*pecificSessions* set up and action requests.

The *Session Manager* implements the core functionalities of the *Wrapper*. It is in charge of (i) receiving and handling *specificSession* configuration data; (ii) managing the collaboration policies as it receives event notifications; and (iii) sending event notifications to other collaborative applications.

[4] JNI (Java™ Native Interface) is used for integrating non Java based collaborative applications.

5.2 Registering a Newly Integrated Application

To register with LEICA, as illustrated in figure 7, the *Wrapper* publishes its services in a *Private UDDI Registry* [17]. In multi-server applications, a *Master Server* is designated to register the application. In P2P applications, registering is made through the *P2P Proxy*.

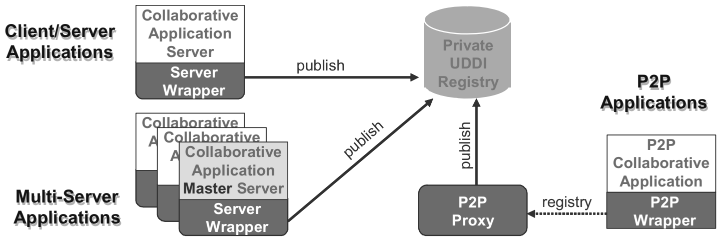

Client/Server Applications
Collaborative Application Server
Server Wrapper
publish
Private UDDI Registry
P2P Applications
Collaborative Application Master Server
Server Wrapper
publish
publish
Multi-Server Applications
P2P Proxy
registry
P2P Collaborative Application
P2P Wrapper

Fig. 7. Registering collaborative applications to LEICA

For implementing the *UDDI Registry*, we use jUDDI [18], a Java implementation that complies with UDDI 2.0. *Wrapper*'s *WS Interface* and the *P2P Proxy* use UDDI4J [19] (Java API for UDDI interaction) to interact with the *UDDI Registry*.

5.3 Creating *SuperSessions*

Figure 8 schematizes the necessary steps for creating a new *SuperSession*.

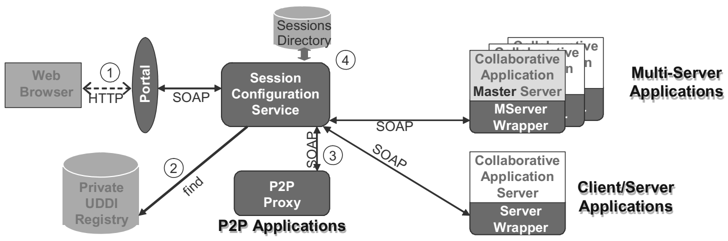

Sessions Directory
4
Web Browser
1
HTTP
Portal
SOAP
Session Configuration Service
Collaborative Application Master Server
MServer Wrapper
Multi-Server Applications
SOAP
SOAP
3
SOAP
2
find
Private UDDI Registry
P2P Proxy
P2P Applications
Collaborative Application Server
Server Wrapper
Client/Server Applications

Fig. 8. Configuration of new *SuperSessions*

1. A Web portal is used to access the *Session Configuration Service* and start the creation process.
2. The *Session Configuration Service* accesses the *Private UDDI Registry* to find out which are the integrated collaborative applications.
3. The *Session Configuration Service* contacts integrated applications to get information about which specific data are needed for configuring *specificSessions*, and which kind of events it can notify and action requests it can treat. Based on this information, *GSMinfo*, *IAinfo* and collaboration policies can be defined.
4. The *SuperSession* configuration file is finally generated and stored.

To implement all these Web services interactions, we use Apache Jakarta Tomcat 5.0 and Apache SOAP 2.3.1.

5.4 Running *SuperSessions*

Figure 9 illustrates how a *SuperSession* is started.

1. A Web portal is used to start a *SuperSession*.
2. The *SuperSession* configuration file is retrieved and parsed. The collaborative applications to be used in this *SuperSession* are identified.
3. The *Session Configuration Service* contacts integrated applications to set up s*pecificSessions* and to send them the collaboration policies of this *SuperSession*.
4. The *Server Wrapper*s of each collaborative application are interconnected through the *Event Notification System*. From this point, Web services are not used anymore.

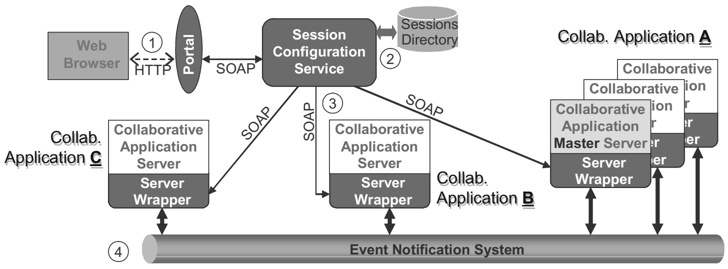


Fig. 9. Starting up a *SuperSession*

In order to implement the event notification system keeping the loosely-coupled nature of LEICA, the publish/subscribe paradigm [2] has been chosen. Publish/subscribe interaction scheme is well-adapted to loosely-coupled environments. In general, subscribers register their interest in patterns of events and then asynchronously receive events matching these patterns, regardless of the events' publishers.

Each *Wrapper* analyses the collaboration policies in order to discover: which type of events it needs to publish, and which type of events it needs to subscribe to. A *Wrapper* just needs to publish *Events* that could enable policy rules, and subscribe to *Events* that could enable a policy rule defining *Actions* to its associated application.

In order to implement the publish/subscribe paradigm for notifying events, we use Scribe [20], a Java based large-scale, peer-to-peer, topic-based publish/subscribe infrastructure. Scribe also provides efficient application level multicast.

5.5 Connecting to a *SuperSession*

In order to connect to a *SuperSession*, a *LClient* application is executed. Figure 10 shows how the connection of a new user is treated.

1. The *LClient* contacts the *Session Configuration Service* and it receives the *GSMinfo*, *IAinfo* and collaboration policies defined to the chosen *SuperSession*.
2. The *LClient* plays the role of a local launch point for P2P and client applications. As it needs to execute the collaboration policies in order to know when P2P/client applications must be launched, it joins the event notification system to receive event notifications.

3. Suppose that, initially, this user is to be connected just to two *specificSessions*, concerning the collaborative applications "B" and "D". Then, the *LClient* runs the client application "B" and the P2P application "D".
4. The *Wrapper* of the P2P application "D" connects to the event notification system and executes the same publishing/subscribing process described in the previous subsection.

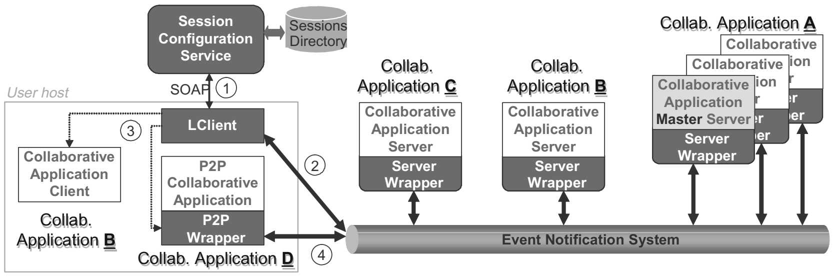

Session Configuration Service
Sessions Directory
User host
SOAP
1
2
3
4
LClient
Collaborative Application Client
Collab. Application B
P2P Collaborative Application
P2P Wrapper
Collab. Application D
Collab. Application C
Collaborative Application Server
Server Wrapper
Collab. Application B
Collaborative Application Server
Server Wrapper
Collab. Application A
Collaborative Application Master Server
Server Wrapper
Event Notification System

Fig. 10. User connection to a *SuperSession*

LClient is a Java application using Apache SOAP 2.3 to contact the *Session Configuration System*, and Scribe's classes to connect to the event notification system.

6 Conclusions and Future Work

This paper has presented LEICA, a loosely-coupled environment for integrating collaborative applications. Existing collaborative applications can be loosely integrated using Web Services as integration technology. In the context of a *SuperSession*, a global collaboration activity is supported where different integrated applications are used in a parallel and coordinated way. Based on the specification of collaboration policies, LEICA defines applications' behavior in response to event notifications.

The current prototype implementation confirms the fact that open source collaborative applications achieve richer interaction levels since we have all the needed flexibility for binding *Wrappers* to applications. However, as new software applications are increasingly coming out of the box with API specifications (sometimes based on Web services), their integration tends to be straightforward.

Regarding Web services, an important effort has been deployed in order to propose solutions for optimizing the transmission and/or wire format of SOAP messages. In this perspective, the current event notification system of the LEICA could also become a Web services-based system without the performance problem actually inherent to SOAP.

Concerning the definition of collaboration policies, no verification of policies' consistency conciseness has yet been performed. Possible solutions based on the use of formal techniques description to guarantee policies' consistency will be studied in a near future.

References

1. Web Services Activity (2005): http://www.w3.org/2002/ws/
2. Eugster, P., Felber, P., Guerraoui, R., Kermarrec, A.-M.: The many faces of publish/subscribe. ACM Computing Surveys, Vol.35. ACM press (2003) 114-131
3. Skype website. http://www.skype.com/
4. Dewan, P.,: An experiment in interoperating heterogeneous collaborative systems. 6th European Conference on Computer Supported Cooperative Work, Copenhagen, Denmark. Kluwer Academic Publishers (1999)
5. Iqbal, R., James, A., Gatward, R.: A practical solution to the integration of collaborative applications in academic environment. 5th International Workshop on Collaborative Editing Systems, hosted by the ECSCW'03, Helsinki, Finland (2003)
6. Spellman, P. J., Mosier, J. N., Deus, L. M., Carlson, J. A.: Collaborative virtual workspace. International ACM SIGGROUP Conference of Supporting Group Work. ACM Press, Phoenix (1997) 197-203
7. Fuchs, L.: AREA: a cross-application notification service for groupware. 6th European Conference on Computer Supported Cooperative Work, Copenhagen, Denmark. Kluwer Academic Publishers (1999)
8. Prinz, W.: NESSIE: an awareness environment for cooperative settings. 6th European Conference on Computer Supported Cooperative Work, Copenhagen, Denmark. Kluwer Academic Publishers (1999)
9. Fox, G. et al.,: A Web services framework for collaboration and videoconferencing. Workshop on Advanced Collaborative Environments, Seattle, Washington (2003)
10. Orfali, R., Harkey, D.: Client/server programming with Java and CORBA. Wiley, NewYork (1998)
11. Distributed Component Object Model (DCOM) (2005): http://www.microsoft.com
12. Roman, E., Ambler S.W.: Jewell T Mastering Enterprise, JavaBeans. Wiley, NewYork (2001)
13. W3C (2004): Simple Object Access Protocol (SOAP). http://www.w3.org/TR/soap
14. Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web Services (chapter 5). In: Web Services – Concepts, Architectures and Applications, Springer-Verlag (2004)
15. Chiu, K., Govindaraju, M. Bramley, R.: Investigating the limits of SOAP performance for scientific computing. 11th IEEE International Symposium on High Performance Distributed Computing. IEEE press (2002)
16. Courtiat, J.P., Santos, C.A.S, Lohr, C., Benaceur, O.: Experience with RT-LOTOS,a temporal extension of the LOTOS formal description technique. Computer Communications, Vol.23, Num.12, July (2000) 1104-1123
17. W3C (2005): Universal Description, Discovery, and Integration (UDDI) http://www.uddi.org.
18. Apache (2005): jUDDI webpage. http://ws.apache.org/juddi/
19. IBM (2005): UDDI4J webpage. http://www.uddi4j.org/
20. Castro, M., Druschel, P., Kermarrec, A.-M., Rowstron, A.: Scribe: A large-scale and decentralized application-level multicast infrastructure. IEEE Journal on Selected Areas in Communications (JSAC), Vol. 20, Num. 8. IEEE press (2002)

Interoperability and eServices

Aphrodite Tsalgatidou and Eleni Koutrouli

Department of Informatics & Telecommunications,
National & Kapodistrian University of Athens, Greece
{atsalga, ekou}@di.uoa.gr

Abstract. eServices are the building blocks for loosely-coupled, distributed applications based on the Service Oriented Architecture (SOA) principles. One of the major benefits they offer is interoperability both between components of service oriented systems and between different systems. Still, the variety and diversity of implementations and interpretations of SOA and the vast amount of emerging standards hinder interoperability. This paper examines interoperability requirements and related issues in the three major eServices categories: Web, Grid and P2P services. Our aim is to provide the basis for a roadmap towards improving interoperability of eServices.

1 Introduction

Service Oriented Architectures (SOA) emerged as an evolutionary step from Object and Component based approaches, with the promise to support the loose coupling of system parts and to provide agility, flexibility and cost savings via reusability, interoperability and efficiency. However, the lack of agreement on what constitutes a SOA and the vast amount of emerging standards makes it difficult to understand and utilize the potentials of eServices technologies. In this context, interoperability, which is one of the basic characteristics and benefits of SOA, needs to be further explored in order to find out the open issues and best practices.

In this paper we present the interoperability requirements and related issues in the eService area, considering interoperability both in terms of intra- and inter- paradigm integration. Our goal is to pinpoint the challenges and provide a roadmap of best practices for interoperability. Thus, in section 2 we describe the model of SOA and the three major trends in eServices, i.e. Web, P2P and Grid services with a focus on the standardization efforts. In section 3 we provide the interoperability issues for each eService area and present synergies and integration efforts, whereas in section 4 we give our concluding remarks which are summarized in a table that provides a general interoperability overview for the three areas of eServices.

2 The eService Model

According to W3C, a Service Oriented Architecture (SOA) specifies a set of components whose interfaces can be described, published, discovered and invoked over a network. eServices are the building blocks of SOA and are mainly instantiated by Web Services (WS), Grid and P2P Services which are briefly described below.

Web Services are self-contained, modular applications, accessible via the Web, that provide a set of functionalities to businesses or individuals. We are currently witnessing the rapid development and maturation of a stack of interrelated standards that are defining the WS infrastructure along with a great number of development tools that support the WS development. The key standards for describing, advertising, discovering and binding WS are WSDL, UDDI and SOAP. Besides, there are ongoing standardization efforts in WS composition, orchestration, choreography, security and management (e.g. BPEL4WS, ebXML, WS-Security, etc.). The current Web Services Protocol Stack, along with details regarding the standardization of the various WS protocols can be found in [10].

The term Grid refers to a system that is concerned with the integration, virtualization, and management of services and resources in distributed, heterogeneous environments. The Open Grid Services Architecture (OGSA) [4] is a significant effort by the Global Grid Forum towards the standardization of protocols and interfaces which integrates Grid and WSs. OGSA was initially materialized by the Open Grid Services Infrastructure (OGSI) [14], and more recently by the Web Services Resource Framework (WSRF) proposal [22]. Currently the efforts of the major industry players are targeted to the support of the Globus toolkit [5].

The term "Peer-to-Peer" (P2P) refers to a class of systems and applications that takes advantage of resources – storage, cycles, content, human presence – available at the edges of the Internet. Standards for P2P technologies have not yet been established. Efforts for defining specifications are made by the P2P Working Group, whereas two standardization initiatives are Jabber [8] and JXTA [9].

3 Interoperability Concerns

In the eServices domain we view interoperability as the ability of systems, applications and services, to communicate, exchange data and files, work together or operate on behalf of one another. In this section we analyze the specific interoperability issues in the areas of Web, Grid and P2P services (intra-paradigm interoperability) and present synergies between different kinds of eServices (inter-paradigm interoperability).

Web Services Interoperability. WS promise universal interoperability and integration by establishing commonly agreed protocols for mutually understanding what a service offers and for delivering this functionality in an implementation independent way. Interoperability of legacy applications is also enabled facilitating a seamless integration between heterogeneous systems. Furthermore, new services can be created and dynamically published and discovered without disrupting the existing environment. Thus, WS technology provides a means of interoperating between different software applications, running on a variety of platforms and/or frameworks.

The issue of WS interoperability is addressed at a *conceptual level* by the W3C′s Web Services Architecture (WSA) [23], which identifies the global elements of the global WS network required to ensure interoperability between WS. The various WS standards and enabling technologies address *technical level* interoperability. The common standards for WS *description, publication and invocation* (WSDL, UDDI, SOAP) have effectively become de facto standards, and, thus, support basic interoperability. However, there is a need for enhanced interoperability in all WS operations. Web Services *description* should include not only functional characteristics of WSs, but also common semantic information that will enable the meaningful interoperation between different WSs. Semantic description is not supported by current WS description standards (e.g. WSDL, whereas ebXML addresses some semantic issues) although some standardized ontology languages, such as OWL [15] and OWL-S [16], can be used for providing WS description semantics. WS *management* also requires common management semantics, in relation to management policies and capabilities, to be understood by the requester and provider entities [23]. Furthermore, the wide range of protocols that have been proposed for WS security, reliability and composition hinder interoperability.

Enhancing interoperability between different implementations of WS technologies is the goal of the Web Services Interoperability (WS-I) organization [24], that defines how existing, stable and widely accepted WS standards should be used. Developers should use implementations of standards that have proven interoperability (examples are the WSDL/UDDI/SOAP initiative and integration of SOAP into ebXML), and also keep up with the advancement of standards.

Grid Services Interoperability. Grid services interoperability can be viewed along two different dimensions: between distributed resources in a Grid application, and between different Grid applications.

Interoperability between different distributed resources in a Grid application is a main goal of the various Grid projects despite the different infrastructures they use and the different aspects on which they focus. The OGSA/OGSI and WSRF models provide a framework for Service Oriented Grids aiming at supporting interoperability of distributed services and resources. Grid middleware implementations based on these models provide services and promote interoperability by allowing interoperation of Grid components independently of the operating system and network topology.

Significant effort has also been channelled towards enabling interoperability between different Grid applications. There is a number of approaches that include the definition of a minimal set of Grid services which enable the interoperation of different Grid applications, the integration of different Grid infrastructures, the common Grid resources description [6] and the Semantic Grid [18] which aims at providing interoperability across time as well as space for reuse of services, information, and knowledge.

P2P Services Interoperability. P2P services interoperability, similarly to Grid Services, can be viewed either as: interoperability between different peers in a P2P network, or between different P2P applications.

Interoperability between different peers needs advanced interoperability techniques, since the various heterogeneous nodes of a P2P network need to communicate, exchange content and aggregate their diverse resources, such as computing power or storage space. Most P2P systems use proprietary implementations and protocols for the peers interoperation and functionality. Enhanced interoperability between heterogeneous peers is supported by *semantic routing* which is recently addressed by researchers, e.g. in Edutella [3].

Interoperability between different P2P applications has not been addressed by early P2P applications, which set up closed peer networks, accessible only to them, whereas, currently, only a few P2P systems can interoperate, such as Magi with JXTA. Efforts towards improved interoperability are made by the P2P Working Group. Early attempts on interoperability include Jabber and Groove which are mainly extensible attempts, and not fully interoperable systems. A common infrastructure that will contain the core services of a P2P application could be a more appropriate approach which is mainly addressed by JXTA and Microsoft's .Net [11].

Inter-paradigm Interoperability. Integration of heterogeneous eServices allows the exploitation of the specific characteristics and benefits of each eServices type by the other eServices, leading to more flexible, efficient and interoperable service-oriented systems. In the following we present synergies between Web, Grid and P2P services and the ways they promote inter-paradigm interoperability in the eServices area.

As already discussed, Grid and WS technologies are in a convergence process, led by the OGSA/OGSI and WSRF proposals. Grid developers can thus exploit the experience of the WS community and concentrate on building the higher-level services that are specific to the Grid application domain. There is a need, however, for WS specifications that could safely be used for building interoperable Web Service Grids, and for this reason a WS specification profile WS-I+ has been proposed in [1]. Another proposal on how Grid applications could be built using existing WS specifications is found in [17].

Recently we are also witnessing a strong movement towards WS and P2P services working in conjunction. Synergies between WS and P2P services include: WS discovery using a P2P-based approach [2], peer discovery in P2P systems using WS as registries, WS interconnection in heterogeneous networks [19], search engines, such as Google, based on P2P and WS, and JXTA projects that incorporate WS.

The techniques that the P2P and Grid models use to handle some of the main issues of distributing computing are discussed in [21] in order to find a common foundation that could alleviate the complexities of each other and fulfill the need for secure, scalable and decentralized collaboration. Another approach to combining aspects of Grid and P2P computing is found in the proposal for a new architecture stack for Grids presented in [13].

The confluence of Web, P2P and Grid services provides the foundation for a common model allowing applications to scale from proximity ad hoc networks to worldwide-scale distributed systems. Some approaches and research projects have started to appear towards supporting this convergence and reusability of the three categories of eServices by providing appropriate models and platforms [12][20].

4 Concluding Summary

In this paper we investigated the interoperability potentials and challenges of WS, P2P and Grid services which are the building blocks of SOA and are known as eServices. Our observations are summarized in Table 1 that offers an overview of the interoperability requirements and existing and possible solutions. We believe that the work presented in this paper can be the basis for a roadmap towards improving interoperability in the eServices area which is one of the main benefits of Service Oriented development.

Table 1. Interoperability approaches and requirements for eServices

		Web services	**P2P services**	**Grid services**
Standardization Efforts	**Common Architecture**	Web Services Architecture (W3C)	Proposed architecture: JXTA	OGSA
	Standards for basic activities (description, publishing, invocation)	WSDL / UDDI / SOAP initiative, ebXML	Standardization efforts: JXTA, Jabber for instant messaging systems, P2Pwg	OGSA/OGSI and WSRF models, no standardised Grid middleware implementations
	Standards for value added activities (e.g. composition, management, security)	Many standardization efforts (BPEL4WS, WS-security) but a few mature standards	No standards	OGSA security architectural components, no standardised implementations
	Semantics support for basic and value-added activities	Partially addressed (e.g. ebXML)	New approaches: Semantic routing (e.g. RDF-based routing algorithms)	Partially addressed
Other approaches	**Approaches to intra-paradigm interoperability**	WS-I and WS-I+ profiles, integration of different standards (e.g. SOAP & ebXML)	Most different P2P systems do not interoperate, exceptions: Jabber with other IM systems, Magi with JXTA, different JXTA -built systems	Need for interoperability between different infrastructures, Grids middleware services enable interoperation independently of network and OS
	Integration Efforts for heterogeneous eServices	• P2P based WS discovery • Peer discovery in P2P systems using WS as registries • Search engines built using WS and P2P technology • WS Grids based on specific WS standards (WS-I+ profile) • Grid architecture based on P2P principles • Convergence of eServices for unified service discovery		

Acknowledgement

This work has been partially supported by the IST Network of Excellence INTEROP [7]. The authors would like to thank the INTEROP researchers George Athanasopoulos and Vangelis Floros for their contribution in earlier drafts of this paper.

References

1. Atkinson, M., DeRoure, D., Dunlop, A., Fox, G., Web Service Grids: An Evolutionary Approach, UK e-Science Technical Report (2004)
2. Banaei-Kashani, F., Ching-Chien Chen, C., Shahabi, C., WSPDS: Web Services Peer-to-peer Discovery Service, Proc. of International Symposium on Web Services and Applications (2004)
3. Edutella project, http://edutella.jxta.org
4. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, Globus Project (2002)
5. Foster, I., Kesselman, C.: The Globus Toolkit, In Ian Foster and Carl Kesselman, editors, The Grid: Blueprint for a New Computing Infrastructure (1999), 259-278
6. Grid Interoperability Project, http://www.grid-interoperability.org
7. INTEROP Network of Excellence http://interop-noe.org/INTEROP/
8. Jabber (2002), http://www.jabber.org/
9. JXTA, http://www.jxta.org/
10. Lawrence, W., The Web Services Protocol Stack (2005), http://roadmap.cbdiforum.com/reports/protocols/
11. Microsoft .NET, http://www.microsoft.com/services/net/default.asp
12. Milenkovic, M., Robinson, S., Knauerhase, R., Barkai, D., Garg, S., Tewari, V., Anderson, T., Bowman, M., Intel, Toward Internet Distributed Computing, published at Computer Society (IEEE), (2003), Vol. 36, No. 5, 38-46
13. Next Generation Grid 2nd Group report (2004), http://www.cordis.lu/ist/grids
14. OGSI, http://www-unix.globus.org/toolkit/draft-ggf-ogsi-gridservice-33_2003-06-27.pdf
15. OWL Overview, W3C Recommendation (2004), http://www.w3.org/TR/owl-features/
16. OWL-S 1.0 Release, http://www.daml.org/services/owl-s/1.1/
17. Parastatidis, S., Webber, J., Watson, P., Rischbeck, WS-GAF: A Framework for Building Grid Applications Using Web Services, Journal of Concurrency and Computation: Practice and Experience (2005), 391-417
18. Roure, D., Jennings, N., Shadbolt, N., The Semantic Grid: Past, Present and Future, to appear in IEEE Proceedings March 2005
19. Schattkowsky, T., Loeser, C., Müller, W., Peer-To-Peer Technology for Interconnecting Web Services in Heterogeneous Networks, Proc. of 18th International Conference on Advanced Information Networking and Applications (2004)
20. SODIUM project, http://www.atc.gr/sodium
21. Talia, D., Trunfio, P.: Toward a Synergy Between P2P and Grids, published in IEEE Internet Computing, July 2003, 94-96
22. The WS-Resource Framework (WSRF), http://www.globus.org/wsrf/
23. Web Services Architecture, http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/
24. Web Services Interoperability Organization, http://www.ws-i.org/

Model-Driven Self-management of Legacy Applications

Markus Debusmann, Markus Schmid, and Reinhold Kroeger

Fachhochschule Wiesbaden - University of Applied Sciences,
Department of Computer Science, Distributed Systems Lab,
Kurt-Schumacher-Ring 18, 65197 Wiesbaden, Germany
`{debusmann, schmid, kroeger}@informatik.fh-wiesbaden.de`

Abstract. Increasing complexity of todays applications and services leads to the emerging trend of self-managing systems. Legacy applications often offer interfaces for manual management and very rarely do provide self-management features. Therefore it is very important to reuse existing management interfaces for achieving self-manageability.

This paper presents our model-driven self-management approach of legacy applications. We introduce a framework for model-driven service level management which transforms abstract SLAs defined in UML into concrete SLA descriptions and deploys them for management. These SLAs are used to define the goals for our self-management agent which is responsible for providing feedback control. Its management knowledge is transformed from UML models using also a model-driven approach.

1 Introduction

The ongoing economic pressure on enterprises increases the need for outsourcing and purchasing services from external service providers. Service Level Agreements (SLAs) [1, 2] represent agreed upon contracts between service providers and service customers. SLAs are an important element within the outsourcing business and define Quality of Service (QoS) parameters of the provided service. Typically, these SLA parameters are related to performance and availability, e.g., the mean value for the response time of service functions. Service Level Objectives (SLOs) define reference values (e.g. thresholds and value sets) for SLA parameters. In order to detect violations, the SLA parameters have to be monitored and compared to the SLOs at runtime.

Over the past years, the complexity of distributed applications has significantly increased, accompanied by a rapid change of emerging middleware technologies, such as CORBA [3], J2EE [4], and Web Services [5]. These frequent technology changes as well as the overall complexity significantly complicate the management of today's distributed applications. Therefore self-managing functionality of applications is becoming more and more crucial. As a consequence, IBM started its *Autonomic Computing Initiative* [6] in 2001 whose goal is to develop self-managing systems covering one or more of the following aspects: self-configuration, self-optimisation, self-healing, and self-protection [7]. IBM's competitor HP also introduced a self-managing approach called *Adaptive Enterprise* [8].

The development of self-managing systems is a very complex task since such a system has to incorporate the knowledge that is usually hidden in the brains of human administrators. This leads to a number of critical research questions, like:

- How can management knowledge be formally represented in an effective way?
- How does a self-managing system handle situations that were not foreseen during the systems design?
- How can the design of self-managing systems be supported?

In the area of software development, the Model Driven Architecture (MDA) represents an approach for effectively designing complex systems as abstract models, which, ideally, are then automatically transformed into application code.

We claim that the principles of the MDA can tremendously benefit the development of self-managing systems. In this paper, we concentrate on achieving self-manageability for traditionally managed legacy systems. This is very important, since enterprises made large investments in existing software systems which are not easily replaceable. Usually legacy applications offer a set of interfaces for their manual management which should also be used for self-management. In addition, a lot of experience on managing existing applications has been gained during their operation. Our approach has the following advantages:

- existing management interfaces are reused, i.e., the application being managed does not need to be modified,
- existing management know-how is leveraged by formalisation.

Within our approach, the use of MDA for obtaining self-management for legacy applications is twofold:

1. We propose the definition of abstract SLA patterns, that can then be transformed and bound to concrete management platforms. These SLA patterns provide the basis for setting up a management platform for monitoring SLA compliance, and for configuring a self-management agent for carrying out corrective actions on the system under management based on the compliance information.
2. The knowledge base of a self-management agent can be set up by using a model driven approach, i.e., abstract management knowledge is transformed into concrete management directives by using appropriate model transformations.

The MDA-based management approach provides the necessary level of abstraction needed for dynamic on-demand services in large-scale heterogeneous environments. Furthermore, the proposed abstract SLA templates provide added value by being reusable and long-lasting.

The remainder of this paper is structured as follows: section 2 presents the foundations of the Model Driven Architecture and the extension mechanisms of the Unified Modeling Language, while section 3 presents our architecture for model-driven self-management in detail. After this, section 4 presents the application of the implemented prototype in a sample e-business scenario. The paper concludes with a summary and an outlook of our future work in section 5.

2 Model Driven Architecture and UML

The design and implementation of modern software systems is complex and costly due to rapid and continuous technology changes. These require tremendous efforts in application integration, as well as support for developing adaptive and open applications that can keep track with the steady technological progress. The Model Driven Architecture (MDA) [9], defined by the Object Management Group (OMG), aims at solving integration and development problems. It defines a model-based methodology which separates the application problem domain from the technological specifics of the underlying computing infrastructure [10].

The MDA distinguishes between three models: the *Platform-Independent Model (PIM)*, the *Platform-Specific Model (PSM)*, and the *Platform-Specific Code (PSC)*. The models can be converted into each other by a process called *model transformation*. Typically, an MDA tool (or a chain of tools) supports the definition and transformation of these models.

The PIM is the core ingredient of the MDA. It specifies a system independent of any platform-specific technology that is later used to implement the system. The PIM focuses on the business functionality and behaviour including the relationships between the components involved. MDA's vision is the specification of computationally complete PIMs. Ideally, all development is carried out at the modelling level, and these models can be exploited for testing and verification purposes.

The PSM is obtained by transforming the PIM for a given target platform, e.g., CORBA, J2EE, or Web Services. The goal is to capture as much of the platform know-how as possible in the PIM-PSM transformation and to generate as much of the PSM as possible automatically. Thus, the developer can concentrate on the application logic in the PIM and is relieved from the complexities of platform-specific code.

The last step is the generation of the (platform-specific) code from the PSM. Ideally, a MDA tool generates all necessary application code (and associated files) for the chosen programming language and deployment platform.

For the specification of PIM and PSM, the MDA proposes the Unified Modeling Language (UML) [11] which is also an OMG standard. UML supports various meta-level mechanisms to extend its specification elements: stereotypes, tagged values, and constraints. A profile represents a set of extension elements and is used for adapting UML to the needs of certain problem domains. For the exchange of UML models, e.g. between UML tools, the OMG has defined an XML-based interchange format called XML Metadata Interchange (XMI) [12].

3 Architecture for Model-Driven Self-management

This section introduces our architecture for model-driven self-management. Section 3.1 gives an introductory overview while section 3.2 discusses the model-driven service level management in detail. After this, section 3.3 describes the structure of our self-management agent and the set up of its knowledge base, using a model-driven approach.

3.1 Overview

Figure 1 depicts a high-level view upon our architecture for model-driven self-management. The architecture comprises two subsystems:

– the *SLM system* for deploying and monitoring SLAs, and
– the *self-management agent* for accomplishing corrective actions.

A prerequisite for an effective self-management is the formal and unambiguous specification of management objectives. This is the task of a service level agreement which specifies all management-relevant SLA parameters of the system under management and the SLOs, the self-management system is trying to fulfil. The service level management (SLM) system is responsible for observing the conformance level of an SLA. Therefore, SLAs are being deployed by the SLA deployment system and then monitored by the SLA monitoring system. The system retrieves raw data and metrics from the system under management and computes the management-relevant parameters which are then compared to the defined SLOs.

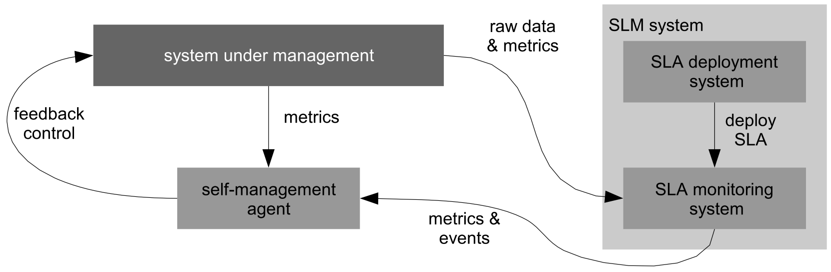


Fig. 1. Overview of the general architecture

In order to achieve self-manageability the self-management agent has to decide whether an adaptation of the system under management is required or not. Its decisions are based on metrics directly retrieved from the system under management and on metrics and events retrieved from the SLM system. These events may signal that a certain SLO is about to be violated and thus a proactive corrective action may be required. For issuing feedback control the self-management agent uses existing management interfaces of the system under management.

The following two sections discuss the SLM system and the self-management agent in more detail.

3.2 Model-Driven Service Level Management

This section describes the concept of the model-driven management of service level agreements realised within the SLM system. The deployment of an SLA is a prerequisite for this.

The central idea of our approach is to apply the principles of the Model Driven Architecture to the problem domain of Service Level Management. Therefore, we need to define the meaning of the different MDA models (PIM, PSM, and PSC) for the area of SLM first. In our approach, the PIM represents an *SLA pattern* which defines an abstract SLA which can be bound to various appropriate configurations and environments. Thus, an SLA pattern only contains the types and their relations to various other component types and an abstract description how SLA parameters are derived.

In the next step, an SLA pattern ("the PIM") is transformed into a concrete SLA ("the PSM"), which we call *SLA instance*, by binding the pattern to concrete configuration information. This includes the binding of the abstract component types to concrete component instances as well as the identification of a concrete management framework, that is used for managing the SLA (e.g. WBEM/CIM). An SLA instance contains all information necessary for configuring the management infrastructure for the given SLA.

The third step is the actual deployment of the SLA instance ("the PSC"). The result of this step is the actual configuration of the management infrastructure enabling it to autonomously manage the conformance of the SLA.

Figure 2 depicts the overall architecture for the model-driven deployment of SLAs.

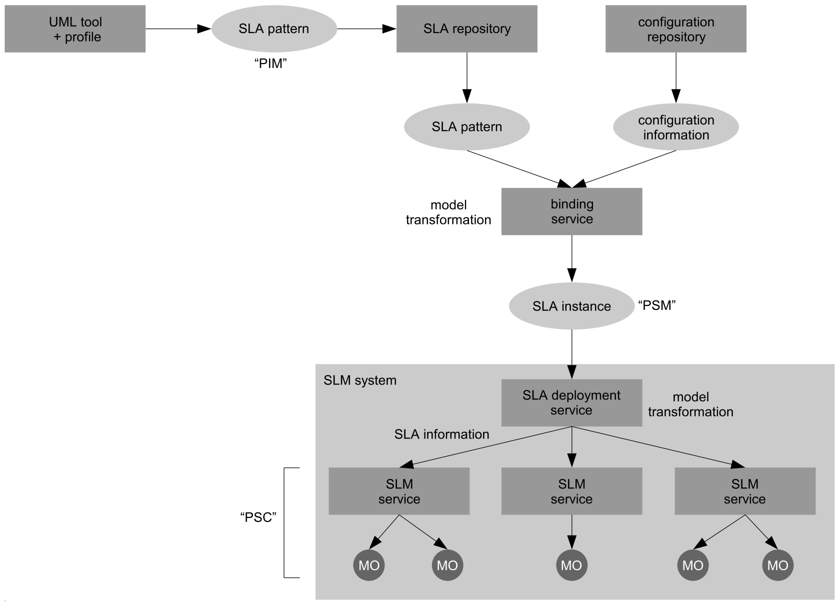


Fig. 2. Architecture for model-driven deployment of SLAs

Defining an SLA Pattern. The starting point for Service Level Management is the signing of a legal contract (SLA) between a service customer and a service provider. In our approach, the formalisation of the contract is done in UML by an administrator using a UML tool and an appropriate UML profile tailored for SLA definition (for details see [13]). The result of the formalisation is the SLA pattern which is stored in the SLA repository for further use. An SLA pattern specifies the relations and dependencies of service types in an abstract manner. For example, a service type for providing HTML content might depend on a web server, a web container, a database, some machines hosting the components, and network connections between them. The virtue of the approach is that for a certain type of service a pattern has to be defined only once and can then be re-used and instantiated multiple times.

Generating an SLA Instance. For actually managing an SLA, a corresponding SLA pattern is retrieved from the SLA repository and bound to a concrete configuration. This is a rather complex task, because all real instances of the types identified in the SLA pattern have to be known. The necessary information is usually retrieved from a configuration repository which contains all relevant information concerning the dependencies between services and components. The transformation of the SLA pattern into an SLA instance is performed by the *binding service* (see figure 2). As explained above, such an SLA instance is constructed for a certain SLM system.

Deploying an SLA Instance. The final MDA step is the generation of platform-specific code. For Service Level Management this corresponds to the physical deployment of the SLA instance generated in the previous phase.

The SLA instance is passed to an *SLA deployment service* (see figure 2) which is specific for the target SLM system. The SLA deployment service configures the services of the SLM system that are responsible for managing the SLA conformance. For example, when using WSLA [14] as SLM system, the WSLA deployment service will configure WSLA's measurement service and condition evaluation service. After configuration, these services are set up to autonomously manage the deployed SLA.

Prototype Implementation. The SLA UML profile has been defined using the UML modelling tool Kase [15]. An example of an SLA pattern modelled using UML can be found in [13]. The models are exported as XMI [12] documents and further processed using the Xalan-J engine (http://xml.apache.org/) and an XSLT [16] style-sheet. The result of this simplification process is an SLA pattern represented in XML and containing only the information that is necessary for representing the SLA.

For instantiating SLA patterns we developed a Java GUI. We used the *Java Architecture for XML Binding* (JAXB, http://java.sun.com/xml/jaxb/) framework for efficiently implementing the access to the XML-represented SLA pattern. The GUI uses a set of plugins for different target platforms to generate the SLA instances. After choosing an SLA template and the desired target platform, the plugin prompts the administrator for the information which is necessary to bind the pattern. The generated instance is then handed over to the platform-specific deployment service for installation. Currently, the SLA patterns are simply read from the file system. In the future, the SLA repository will be implemented as a Web Service.

3.3 Model-Driven Generation of Management Knowledge

We will now describe the model-driven generation of management knowledge which is subsequently used by the self-management agent.

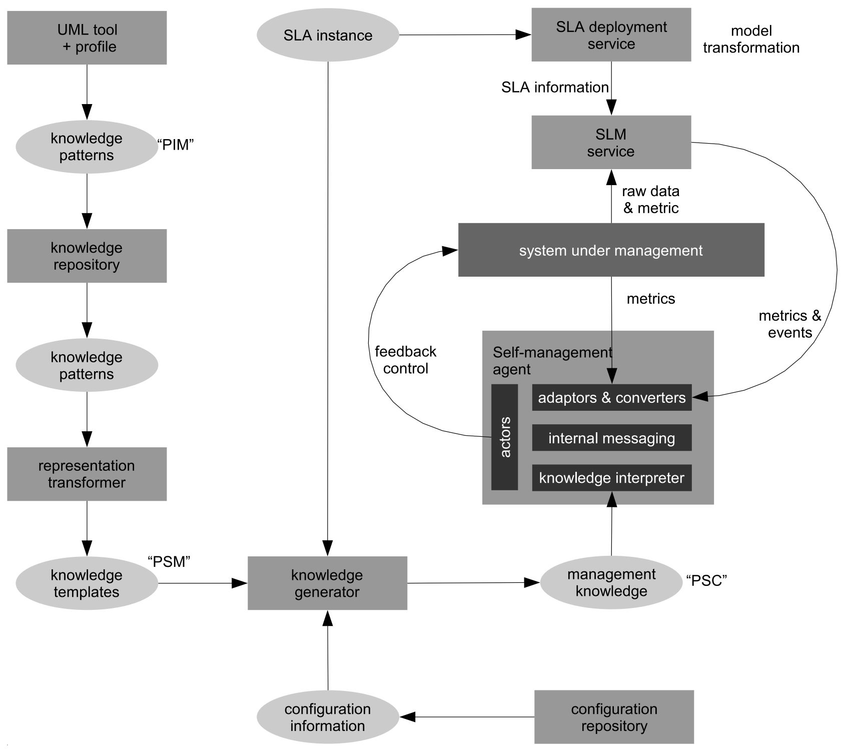


Fig. 3. Architecture for model-driven configuration of the Self-Management Agent

Figure 3 depicts the architecture for the model-driven configuration of the self-management agent which is tightly interrelated with the architecture for the model-driven deployment of SLAs described in the previous section.

Structure of the Self-management Agent. Internally, the self-management agent consists of four major building blocks. Details on the architecture of the self-management agent are provided in [17]. Adaptors and converters are responsible for retrieving metrics and receiving events. They are also responsible for converting this data into the internal messaging format. The task of the internal messaging subsystem is the delivery of all internal information and events. The knowledge interpreter in combination with the management knowledge is the 'brain' of the self-management agent. The management knowledge is evaluated on regular intervals and on the occurrence of certain events. Necessary corrective actions are carried out by application-specific *actors* which realise the feedback control. In a simple scenario actors could be implemented by operating system commands for killing and restarting processes. Actors could also provide a generic interface to interact with a certain class of legacy applications, e.g. an SNMP interface to communicate with SNMP-enabled applications.

Currently, we assume that the person modelling the management knowledge knows or anticipates the correlation between the system metrics and its control variables. Since we focus on achieving self-manageability for legacy applications the necessary knowledge should exist. For new applications partial knowledge can be already gained in the development process. This knowledge is then being gradually completed.

Prototype Implementation. The self-management agent was implemented on AIX and Linux, using C++. The two main design goals for the agent are modularity and high availability. All functionality of the self-management agent (the adaptors & converters, the knowledge interpreter, and the actors) are implemented as shared libraries which can be dynamically loaded into the agent at runtime. The core of the agent is the *module manager* which provides a framework for concurrent communication between the modules.

For achieving high availability the self-management agent is realised as a self-checking process pair. The agent continuously self-diagnoses its modules in order to ensure proper operation. In order to be able to manage a wide range of applications the self-management agent supports a large number of adaptors and converters as well as actors. At the moment, we support SNMP including traps, log analysis, shell scripting, CORBA, WBEM/CIM, and Web Services.

Currently, the self-management agent only supports a simple policy language (defined as an XML schema) for describing the management knowledge, but the architecture is open to support other paradigms, such as finite state machines or neural networks.

Modeling Management Knowledge. Similar to the abstract nature of the SLA patterns, the management knowledge of the self-management agent is described in an abstract manner, called *knowledge patterns*. As a starting point, we use UML state-charts and UML activity diagrams in conjunction with an UML modelling tool and an appropriate profile for describing the knowledge patterns. After modelling, these patterns are stored in a knowledge repository for further use.

For instantiating management knowledge appropriate knowledge patterns have to be retrieved from the knowledge repository. In a first step, the patterns (the "PIM") are transformed by the *representation transformer* into a desired knowledge representation. From our point of view there are certain limitations related to possible target formats of knowledge. We expect to be able to successfully transform our knowledge patterns into policy languages, logical expressions or certain finite state machine models. For different representation formats, such as AI rules, an alternative modelling approach is required.

The result of the transformation process are *knowledge templates* (the "PSM") which still represent abstract knowledge but already for a target representation format. Finally, the *knowledge generator* generates concrete *management knowledge* (the "PSC"). The generator instantiates the knowledge templates using the information contained in the SLA instance generated by the SLM system (see section 3.2). Possibly, additional configuration information retrieved from a repository has to be included as well. The generated management knowledge is necessarily application-specific, but may in large parts be similar for certain application types, e.g. Web Servers. It is interpreted by the self-management agent for deciding whether feedback into the system under management is required and which (application-specific) actors are to be invoked in this case.

3.4 Related Approaches

[18] describes an approach which successfully utilises MDA for modelling QoS aspects when designing distributed applications.

IBM carries out significant research work on self-management, but the design of the controllers presented is highly application-specific: [19] describes the application of control theory to a DB/2 DBMS, [20] shows ways to optimise queueing in a Lotus Notes server. In principle, such complex controller-approaches could complement our architecture, however a possible interaction with SLAs and generated management knowledge requires further analysis.

4 Sample Application Scenario

The prototype has been evaluated in our e-business application infrastructure which represents the system under management (see figure 4). This infrastructure consists of the Squid Web proxy on the client side and the Apache Web server, the Tomcat Web container, the JBoss application server, the ORBacus CORBA platformand the relational database MySQLon the server side. Our environment is fully instrumented using the OpenGroup *Application Response Measurement (ARM)* API [21] for determining the response and processing times of individual components within our e-business environment.

The collected performance measurement data is used by the SLM system for validating SLOs (see [22] for details). In our sample scenario we defined SLOs concerning response time and availability. The actual response times are monitored by the SLM system. In case a response time SLO is violated an appropriate event is sent to the self-management agent in order to trigger a corrective action. The self-management agent tries to fulfil the response time goals by starting additional JBoss instances within a cluster. The EJB client within Tomcat uses the JBoss SmartProxy mechanism for achieving load balancing. If the response times decreases below a certain limit, the self-

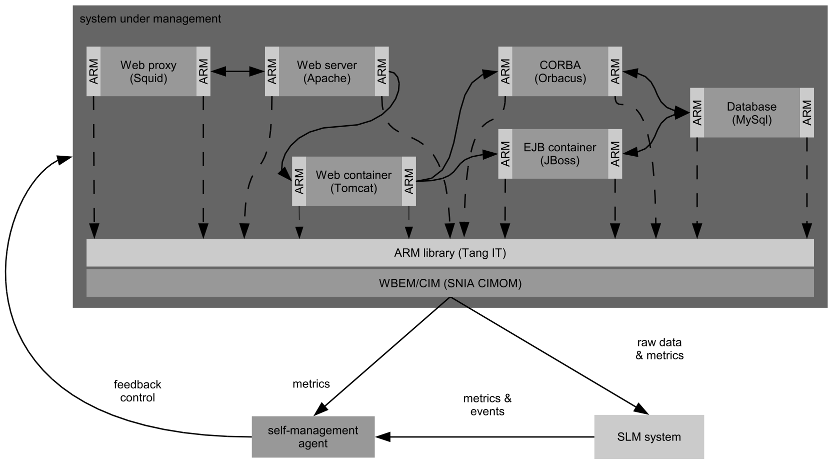


Fig. 4. Managing an e-business environment

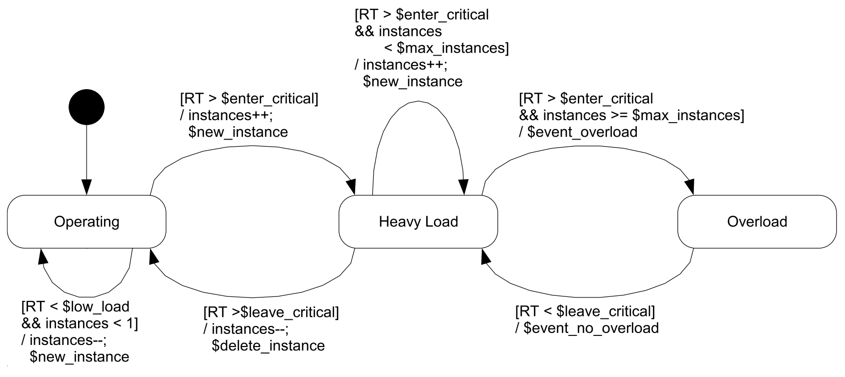


Fig. 5. Knowledge pattern for managing JBoss instances

management agent again gradually removes JBoss instances from the cluster in order to save resources.

The availability of the components is observed by the self-management agent itself via its adaptors and converters. In case a component crashes it is restarted automatically by the self-management agent.

Figure 5 depicts the knowledge pattern for managing the number of JBoss instances. The pattern is modelled as a state machine consisting of three states. The automaton initially starts in the `Operating` state and remains there as long as the response time requirements are satisfied. If the response time rises above a certain limit (`enter_critical`) and the maximum number of parallel JBoss instances has not been reached, a new instance is started and the state machine enters the `Heavy_load` state. As long as the response time remains above `enter_critical` the automaton continues to start new instances until the maximum number is reached. If the response time increases further and the maximum number of instances is already running, the state machine sends the `event_overload` notification and enters the `Overload` state. The notification indicates that additional intervention is required by a superior self-management agent or by a human administrator. If the response time drops again below `leave_critical`, the state machine will send an `event_no_overload` notification and will switch back to the `Heavy_load` state. If the response time drops further below `leave_critical`, the automaton will remove instances in order to save resources and will reenter the `Operating` state. If the response time falls below `low_load`, the state machine will continue removing instances.

Elements starting with `$` represent placeholders that are replaced by the knowledge generator during the transformation process when creating the management knowledge. As can be seen, the presented knowledge pattern is not limited to managing JBoss instances but can be reused for response time management of any component type where multiple instances reduce the resulting response time.

5 Conclusions and Future Work

The constantly increasing complexity of distributed applications and services requires a higher degree of self-management in order to efficiently fulfil rising quality requirements. Traditionally, most applications solely provide interfaces for manual management. In this paper, we propose an approach for achieving self-manageability of legacy applications by leveraging their existing management interfaces. Our approach uses principles of the Model Driven Architecture for obtaining self-management. The overall architecture consists of two basic subsystems: the SLM system for deploying and monitoring SLAs and the self-management agent for realising feedback control.

For effectively deploying SLAs in order to monitor them with an SLM system we propose a model-driven approach. SLAs are modelled in an abstract manner using UML and are called SLA patterns. They can be reused by binding them to various appropriate configurations. A bound SLA pattern is referred to as SLA instance which contains all necessary information for monitoring the SLA. After the deployment of the SLA instance the SLA is automatically monitored. The SLA determines the reference values that should be satisfied by the self-management.

The self-management agent is responsible for achieving conformance with the deployed SLAs. The modular agent consists of adaptors & converters for retrieving metrics and events, an internal messaging system, a knowledge interpreter for the self-management algorithms, and actors for realising feedback control.

The management knowledge is transformed using the model-driven approach as well. First of all, abstract knowledge patterns are modelled as UML state-charts and activity diagrams using an UML tool plus an appropriate UML extension. The knowledge patterns can then be transformed into knowledge templates that represent a desired target representation of the knowledge patterns. Finally, the knowledge templates are instantiated using SLA instances and additional configuration information. The generated management knowledge represents the self-management algorithms that are executed by the knowledge interpreter.

We successfully implemented the model-driven approach for SLA deployment and monitoring and the self-management agent. The implementation of model-driven generation of the management knowledge is ongoing work. Our future work will concentrate on completing the implementation and gaining experience in different modelling formats for knowledge patterns and their transformation into various target formats in order to realise different types of feedback algorithms.

References

1. Lewis, L.: Managing Business and Service Networks. Kluwer Academic Publishers (2001)
2. Sturm, R., Morris, W., Jander, M.: Foundations of Service Level Management. SAMS Publishing (2000)
3. Object Management Group: Common Object Request Broker Architecture: Core Specification. (2004) Version 3.0.3 - Editorial changes, OMG document formal/04-03-01.
4. Sun Microsystems, Inc.: Java 2 Platform Enterprise Edition Specification, v1.4. (2003) Final Release, http://java.sun.com/j2ee/j2ee-1_4-fr-spec.pdf.
5. World Wide Web Consortium: Web Services Architecture. (2004) W3C Working Group Note, http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/.
6. Horn, P.: Autonomic Computing — IBM's Perspective on the State of Information Technology. IBM Corporation. (2001) http://www.research.ibm.com/autonomic/.
7. Kephart, J.O., Chess, D.M.: The Vision of Autonomic Computing. IEEE Computer Magazine (2003) 41–50

8. Hewlett-Packard: Building an adaptive enterprise — Linking business and IT. (2003) http://h30046.www3.hp.com/solutions/utilitydata.html.
9. Object Management Group: Model Driven Architecture (MDA). (2001) OMG document ormsc/2001-07-01.
10. Object Management Group: Model Driven Architecture (MDA). (2001) document number: ormsc/2001-07-01.
11. Object Management Group: OMG Unified Modeling Language Specification. (2003) Version 1.5, OMG document formal/03-03-01.
12. Object Management Group: XML Metadata Interchange (XMI) Specification. (2003) Version 2.0, OMG document formal/03-05-02.
13. Debusmann, M., Geihs, K., Kroeger, R.: Unifying Service Level Management using an MDA-based Approach. In Boutaba, R., Kim, S.B., eds.: Proceedings of the 9^{th} International IFIP/IEEE Network Operations and Management Symposium (NOMS 2004), IEEE (2004) 801–814 Seoul, South Korea.
14. Keller, A., Ludwig, H.: The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. Journal of Network and Systems Management, Special Issue on "E-Business Management" **11** (2003)
15. Weis, T.: Kase UML Tool. (2003) QCCS Project, http://www.qccs.org/KaseTool.shtml.
16. World Wide Web Consortium: XSL Transformations (XSLT). (1999) Version 1.0.
17. Debusmann, M., Kroeger, R.: Widening Traditional Management Platforms for Managing CORBA Applications. In Zieliński, K., Geihs, K., Laurentowski, A., eds.: Proceedings of the 3^{rd} IFIP WG 6.1 International Working Conference on Distributed Applications and Interoperable Systems (DAIS'2001), IFIP, Kluwer Academic Publishers (2001) 245–256 Krakow, Poland.
18. Weis, T., Ulbrich, A., Geihs, K.: Modellierung und Zusicherung nicht-funktionaler Eigenschaften bei Entwurf, Implementierung und zur Laufzeit verteilter Anwendungen. In Irmscher, K., Faehnrich, K.P., eds.: Kommunikation in Verteilten Systemen (KiVS), GI/ITG, Springer (2003) 119–130 (in German).
19. Diao, Y., Eskesen, F., Froehlich, S., Hellerstein, J.L., Spainhower, L.F., Surendra, M.: Generic Online Optimization of Multiple configuration Parameters With Application to a Database Server. In: Proceedings of the fourteenth IFIP/IEEE Workshop on Distributed systems: Operations and Management (DSOM 2003). (2003)
20. Parekh, S., Gandhi, N., Hellerstein, J., Tilbury, D., Jayram, T., Bigus, J.: Using Control Theory to Achieve Service Level Objectives In Performance Management. In: Proceedings of the Seventh International Symposium on Integrated Network Management (IM). (2001)
21. The Open Group: Systems Management: Application Response Measurement (ARM). (1998) Open Group Technical Standard, Document Number: C807.
22. Debusmann, M., Schmid, M., Schmidt, M., Kroeger, R.: Unified Service Level Monitoring using CIM. (2003) EDOC 2003, Brisbane, Australia.

Model-Driven Methodology for Building QoS-Optimised Web Service Compositions

Roy Grønmo[1] and Michael C. Jaeger[2]

[1] SINTEF ICT, P.O.Box 124 Blindern N-0314 Oslo, Norway
`Roy.gronmo@sintef.no`
[2] TU Berlin, FG FLP SEK FR6-10, Franklinstrasse 28/29 D-10587 Berlin, Germany
`mcj@cs.tu-berlin.de`

Abstract. As the number of available Web services increases there is a growing demand to realise complex business processes by combining and reusing available basic Web services. For each of the needed basic Web services there may be many candidate services available from different vendors and with different Quality of Service (QoS) values. This paper proposes a model-driven methodology for building new Web service compositions that are QoS-optimised. We investigate if UML is suitable for modelling the QoS aspects and we explain how transformations can be used to automate most parts of the methodology. Some of these transformations are already implemented in previous work.

As part of the methodology we present a control flow pattern approach that optimises the QoS values of the composition given user-defined requirements and preferences. The pattern approach is compared towards the local and global approaches identified by other authors. Experiments are carried out within our simulation tool that simulates the selection of services and aggregation of QoS in a given composition. The test results are used to identify recommendations of when to use the different approaches based on measurements of computation time of the optimisation component and achieved QoS values for the new composition. The methodology is explained by relating it to a gas dispersion emergency case.

1 Introduction

A growing number of Web services are implemented and made available internally in an enterprise or externally for other users to consume. Such Web services can be reused and composed together in order to realise larger and more complex business processes. We define Web services to be services available by using Internet protocols such as HTTP and XML-based data formats for their description and invocation. A Web service composition is a description of how basic Web services can interoperate in order to accomplish a larger task. There is not yet any de-facto standard for defining Web service compositions although there are several evolving proposals (BPEL4WS [3] etc). All of these composition languages, as well as the other adopted Web service specifications (SOAP, WSDL) use low-level XML code as the representation language. We propose to use a model-driven methodology that provides a relation between the XML specifications and graphical, higher-level models using the Unified Modelling Language (UML). This improves the development and maintenance of new Web services.

In a composition there are many sub-tasks that need to be solved by calls to existing services. There may be many alternative services that can accomplish the same task, but with different QoS offerings such as price and data quality. This means that a selection among these services must be taken. For the automatic selection of services and computation of overall QoS values, we introduce a control-flow pattern approach. Two main hypotheses are elaborated in the paper:

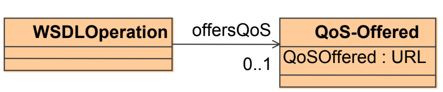
- The forthcoming UML Profile for modelling QoS from the Object Management Group (OMG) [9] can be used to capture the necessary QoS aspects in UML for Web service composition.
- The pattern approach to QoS-optimisation achieves improved overall QoS values of the composition and/or improved computation time compared to previously published approaches.

Both hypotheses are evaluated against a gas dispersion emergency case. The modelling constructs of OMG's UML profile is applied to this case to see if they are suitable and cover all the aspects relevant in this case. The pattern approach and the alternative approaches are simulated by a tool that compares the achieved overall QoS values of the whole composition and the computation time of the algorithms. The tests use the fixed composition of the gas dispersion case and randomly generated candidate services for each task in the composition with randomly assigned QoS values.

This paper is organised as follows. Section 2 provides an overview of the methodology. Sections 3 to 6 go through the four methodology steps in detail as applied to the gas dispersion case. Section 7 discusses assumptions and potential shortcomings in our current methodology. Section 8 describes a simulation tool that is used to compare our optimisation approach with two other approaches. Section 8 presents related work and section 9 summarises with conclusions and by identifying future work.

2 Methodology

This section describes a methodology for designing and implementing QoS-optimised Web service compositions. The methodology is model-driven using UML as the modelling language. We propose to define and implement fully automated transformations whenever applicable. The main focus in our methodology is to choose the most appropriate web services based on their QoS offerings. Therefore QoS offered values need to be retrieved somehow by the service requester. Since there is no standardised way to do this, we shortly explain three alternative ways such information may be retrieved:



```
<operation name="getWind" offersQoS="QoSOfferedURL">
        <input message="tns:getWind"/>
        <output message="tns:getWindResponse"/>
</operation>
```

Fig. 1. Extending WSDL to provide QoS information

- Extending WSDL with a reference to offered QoS (Figure 1). This enables each WSDL operation to be associated with a QoS offerings document which may be updated regularly by the service provider if the values change.
- The service requester negotiates a QoS contract with the service providers. Signed bilateral contracts define the QoS offerings that the service provider is obliged to deliver. The negotiation process may be automatic or manual.
- Asking QoS testing services. A QoS testing service will calculate QoS offered values by running test scenarios against the services, or it may gather information based on reports from other consumers of a service.

Especially with the first and last of these alternatives it will be a question of reliability of the retrieved QoS values. It is outside the scope of this paper to further investigate these alternatives. In the remainder of this paper we assume that the QoS values may be retrieved and be relied upon, and that the WSDL extension mechanism is used. Services without retrievable QoS values must be ignored as candidate services. Note that the QoS requirements will refer to a set of QoS characteristics, and QoS values for the same characteristics need to be specified in the offered QoS of the candidate service.

The methodology has four steps (Figure 2); In step 1 the user designs a composite Web service with a set of QoS requirements; In step 2 an optimisation service is used to select the best suited candidate services for the composition; In step 3 the model is finalised with details of chosen services and computed QoS values for the composition; In step 4 the model is exported into specifications that are used to implement, deploy and publish the composite Web service with its offered QoS values.

We will explain the methodology by applying it to a concrete example. The gas dispersion emergency case of the ACE-GIS project [13] has been extended with QoS aspects. The case is about how to handle an emergency situation caused by an explosion in a chemical factory, with subsequent leakage of poisonous gas. In order to make efficient evacuation plans, the crisis management needs a forecast of the gas plume

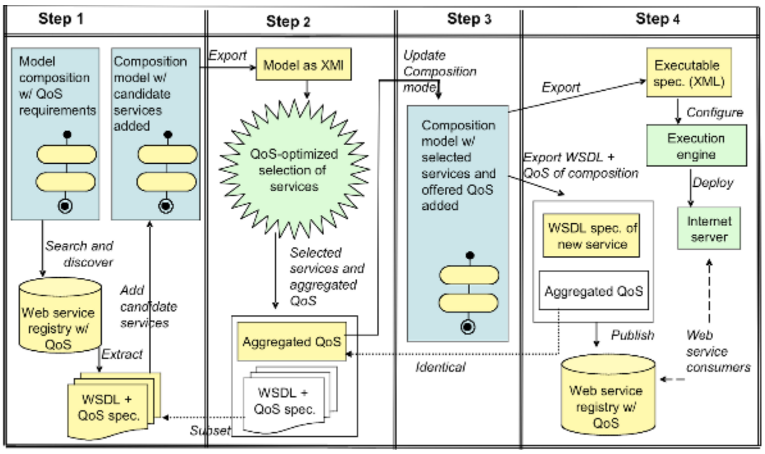

Step 1
Step 2
Step 3
Step 4
Model composition w/ QoS requirements
Composition model w/ candidate services added
Search and discover
Web service registry w/ QoS
Add candidate services
Extract
WSDL + QoS spec.
Export
Model as XMI
QoS-optimized selection of services
Selected services and aggregated QoS
Aggregated QoS
WSDL + QoS spec.
Subset
Update Composition model
Composition model w/ selected services and offered QoS added
Identical
Export
Executable spec. (XML)
Configure
Execution engine
Deploy
Internet server
Export WSDL + QoS of composition
WSDL spec. of new service
Aggregated QoS
Publish
Web service registry w/ QoS
Web service consumers

Fig. 2. Methodology

dispersion. Hence a Web service needs to be designed which calculates the gas plume and displays it on a map. We introduce some QoS values for the different services in the case, but these are only provided as examples and are not grounded by any actual measurements.

3 Step 1: Modelling the Composition with QoS Requirements

This section shows how UML can be used to model QoS requirements within a Web service composition. The QoS modelling builds upon OMG's forthcoming UML profile for QoS modelling [9].

Modelling the Interface and Behaviour. The first step is the interface modelling which identifies the new operations of the new composite Web service. For the gas dispersion example we wished to develop a Web service with one operation: CreateGasDispersionMap(plantID: string, emissionRate: float): base64Binary. By calling this operation with a plant identifier and an emission rate of the gas dispersion, the result will be an image map displaying the expected gas dispersion after a given time period on a background area map. For the identified operation we need to model the behaviour. This can be achieved by using UML activity models that capture the control flow and data flow aspects. Figure 3 shows a simplified activity model of the control flow for the CreateGasDispersion operation. First we get the location of the plant with gas leakage, and then the nearest airport is located since this is required by the Get Wind service[1]. In parallel a gas dispersion plume is calculated and a background map is retreived. Finally, a map with the gas dispersion plume overlaid on the background map is created.

Modelling QoS Characteristics. The QoS concepts need to be precisely defined and used by all the parties involved. The OMG profile uses QoS characteristics to define collections of QoS concepts with precise semantic meaning. Each QoS characteristic contains a set of QoS dimensions with a name and allowed value domain. A QoS dimension also has a direction which is defined as either increasing or decreasing, where increasing means that higher values are preferred. The QoS characteristics for the gas dispersion case are shown in Figure 4. A price is either given as monthly subscription or per call, and the currency is Euro. Execution time is measured in seconds and only with respect to worst case. A user rating specifies the user satisfaction on a scale from 1 to 10 where higher values are preferred. The encryption level ranges from 1 to 4 where higher values are preferred. These levels should be further detailed in order to be precisely defined.

Modelling QoS Requirements. When the QoS characteristics are defined, they can be used to model the QoS requirements. A QoS requirement identifies restrictions on a service with respect to the value domain of specific QoS characteristics. These requirements need to be fulfilled in order to achieve a successful binding to a service. The

[1] In the gas dispersion case, no Web service was found to provide a wind estimate for an arbitrary location. However, a Get Wind service taking an airport location as input, was found.

requirements may apply to a single service within the composition or to the service composition as a whole.

We propose to extend the OMG profile with QoS interests, which is the set of QoS characteristics that are desirable to be optimised. The QoS interests follows the same notational principles as the other parts of the OMG profile, such as the requirements modelling. A possible strategy is to implicitly include all of the QoS characteristics, referred by QoS requirements, as part of the interests set. In many cases this may not be desirable since optimization deals with trade-offs. Any additional characteristic that one tries to optimise may worsen the optimisation of the other characteristics. We instead recommend explicitly modelling all of the characteristics to be optimised within the interests set. The result may then be that characteristics specified only in the requirements set and not in the interests set will not be optimised any further beyond fulfilling the requirements upper and lower bounds. Furthermore we propose that the different characteristics in the interests set are weighted to indicate the relative importance of the interest.

Figure 3 shows the composition of the gas dispersion case with QoS requirements. The QoS requirements in the example (top left note) apply to the composition as a whole, since they are not attached to any the individual services (this is our notational convention). The requirements state that the required price per invocation must be at most 50 Euros and the execution time must never exceed 20 seconds. In addition we specify QoS interests (lower left note) indicating which QoS characteristics we want optimised. Each of these interests has assigned a weight between 0 and 1, where 1 indicates highest importance. The QoS characteristics used are already defined by a separate UML package as shown in Figure 4.

Model with Candidate Services. When searching for services we have ideally discovered candidate services for all the different tasks in our composition model. If not, two possible actions must be taken. Either the composition must be changed or the missing service must be implemented by ourselves. Each Web service is defined syntactically by the de-facto standard WSDL. Updating the composition model with the candidate services can be supported by reverse transformations from WSDL to UML [5]. We also

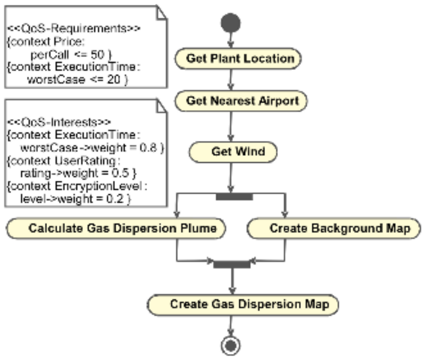
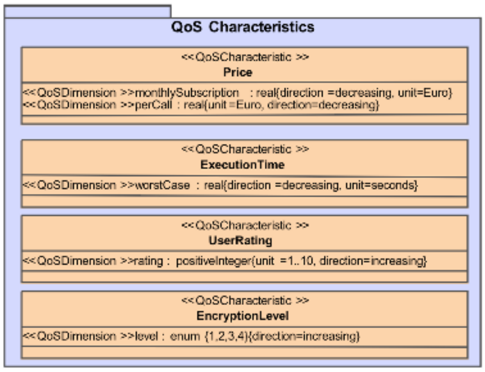

```
<<QoS-Requirements>>
{context Price:
    perCall <= 50 }
{context ExecutionTime:
    worstCase <= 20 }
```
```
<<QoS-Interests>>
{context ExecutionTime:
   worstCase->weight = 0.8 }
{context UserRating:
   rating->weight = 0.5 }
{context EncryptionLevel:
   level->weight = 0.2 }
```

Fig. 3. Modelling QoS Requirements



```
<<QoSDimension >>monthlySubscription : real{direction =decreasing, unit=Euro}
<<QoSDimension >>perCall : real{unit =Euro, direction=decreasing}
<<QoSDimension >>worstCase : real{direction =decreasing, unit=seconds}
<<QoSDimension >>rating : positiveInteger{unit =1..10, direction=increasing}
<<QoSDimension >>level : enum {1,2,3,4}{direction=increasing}
```

Fig. 4. Modelling QoS Characteristics

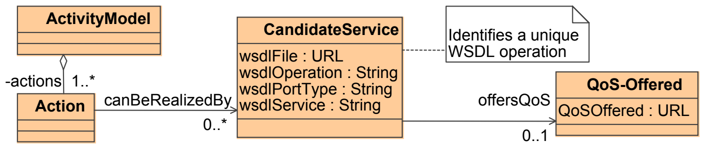


Fig. 5. Metamodel for Candidate Services

need to attach these candidate services to its belonging task. The metamodel for attaching candidate services to the tasks are given in Figure 5. Each task can be realised by many candidate services. A candidate service is defined by a WSDL operation. Note that one WSDL file may provide several operations. The four attributes wsdlFile, wsdlOperation, wsdlPortType and wsdlService will together give a unique reference to the WSDL operation. The WSDL operation refers to a document describing its offered QoS.

The modelling of candidates in UML can be handled by letting each action have a tagged value, CandidateServices, which defines the URL of an XML file with a list of the candidate WSDL operations that can perform the task. Since there may be many candidate services for a task, this list structure is more technically feasible to capture in an XML file than in the UML model itself. The production of such an XML file can preferably be supported by a tool. The composition model with candidate services are sent to the QoS-optimisation algorithm discussed in the next section. The outcome of applying the algorithm will be one selected service for each task in the composition.

4 Step 2: QoS-Optimised Selection

In order to select candidates for tasks in a composition based on QoS requirements, a method is necessary to determine the QoS of a composition depending of the offered QoS of potential candidates. In a preceding paper we have introduced a set of seven structural elements – called *composition patterns* – to model a composition [7]. The composition patterns were derived from a set of workflow patterns, which have been introduced by van der Aalst et al. [12]. Workflow patterns form a set of functional and structural requirements for workflow management systems. We have chosen the workflow patterns as the foundation for the composition patterns because existing flow languages for the composition of Web services are based on a similar approach like modelling languages for workflows. Van der Aalst has also shown that the workflow patterns apply to existing composition languages [11]. The patterns are basic sequential and parallel arrangements of tasks and thus can be seen as the atomic elements of a composition:

Sequence of service executions. A sequence can be ordered or in arbitrary order. For the aggregation model, the order of the service executions is not relevant.

Loop. The execution of a service or a composition of services is repeated for a certain amount of times.

XOR split followed by a XOR join. From a parallel arrangement only one task is started. Thus, the synchronising operation only waits for the started task.

AND split followed by an AND join. From a parallel arrangement all tasks are started, and all tasks are needed to finish for synchronisation.

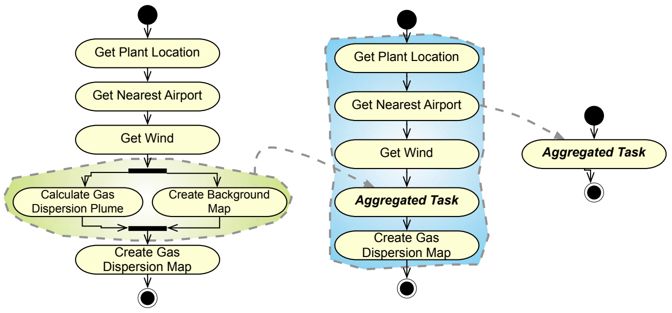


Fig. 6. Example of Collapsing the Graph for Aggregation

AND split followed by a m-out-of-n join. From a parallel arrangement all tasks are started, but less than all tasks are needed to finish for synchronisation.

OR split followed by OR join. In a parallel arrangement some of the available tasks are started and all of the started tasks are needed to finish for synchronisation.

OR split followed by a m-out-of-n join. In a parallel arrangement some of the available tasks are started but only a subset needed to finish for synchronisation.

The composition patterns anticipate that the described flow can be represented using directed graphs that specify the order in which activities are executed. For this example this means that the XMI output, representing the UML model, is transformed into an internal graph representation that consists of the pattern elements. Then for aggregation, we propose to stepwise collapse the graph into a single node by alternately aggregating the sub-patterns starting with sub-patterns that do not contain any further sub-patterns. For the example shown in Figure 6, it means that two aggregation steps are performed: first the parallel arrangement and then the sequence containing the aggregated result from the parallel arrangement is aggregated. This results in a single QoS statement for each of the different QoS characteristics. In summary, this approach enables us to view the aggregation in a pattern perspective, i.e., not the whole composition is relevant at once for the aggregation process, but rather only the local composition patterns, which are accordingly differentiated into sequential and parallel service executions. To cover the OR splits, an aggregation model must know which paths are taken into account for this split. To determine the aggregation of the QoS values, all combinations of possible splits must be taken into account.

Worst case execution time. For the sequential patterns the execution times are added, in the parallel cases the largest value is relevant for giving an estimation about the possible worst case.

Price. For the aggregation of the price in parallel arrangements, every started task is relevant. For parallel split cases, the maximum price possible is relevant.

User rating. The user rating can be aggregated by two approaches. Calculating the mean of the given values within an element performs the aggregation of the rating.

Encryption. The encryption case is very straightforward and considers that for a composition the chosen candidate that is offering the weakest encryption is relevant for the whole composition.

To apply the introduced aggregation rules and the aggregation scheme using the pattern-perspective approach some assumptions must remain valid, which we will discuss in section 7. With pattern-wise aggregation a method is available to calculate the QoS of the whole composition or its parts. Now, this method can be applied in a selection process that needs to evaluate the resulting QoS for different combinations of assigning candidates to tasks.

Pattern-wise Selection. In the selection process, for each task an available candidate is assigned. The simplest approach, called local-based, is to select a candidate by comparing the QoS among all candidates for a particular task and chose the candidate that provides the best QoS. This selection algorithm shows a greedy behaviour. However, the QoS for a composition can be increased if an algorithm would perform the selection by considering the QoS of candidates for other tasks as well: consider our setup in the gas dispersion example. In this example, the parallel execution of the two tasks Calculate Gas Dispersion Plume and Create Background Map is given. Let the optimisation goal be to find the quickest execution while trying to keep the lowest price. Also, choosing a quicker service is usually more costly. For the case that even the quickest candidate for the left task executes longer than any of the candidates for the right task, the optimal assignment for the right task is the cheapest service, regardless how long its execution would take. Due to its nature, this kind of optimisation cannot be performed with a local-based approach.

To address this particular case, our selection algorithm evaluates the candidate services by evaluating the possible assignments within each pattern by performing the following steps:

1. In the graph of composition patterns, the algorithm walks by a recursive approach into the structure and identifies pattern elements that do not contain any sub-patterns. In the example this would be the element with the parallel arrangement.
2. For all tasks within this element, all combinations of candidate assignments are evaluated. For each combination the QoS is aggregated using the pattern-based aggregation and the combination that delivers the best overall QoS is chosen.

To evaluate the combinations among the elements the algorithm compares the QoS of either candidates or aggregated patterns by calculating a ratio between two sets of services. For this the *Simple Additive Weighting (SAW)* can be applied, which was introduced in the context of *Multiple Criteria Decision Making (MCDM)* [6]. By this approach for each QoS value a weight is applied and then a score is calculated for each candidate. The score is determined by firstly normalising the values among the candidates to compare. Let s_{yx} be a single value with x denoting the index of the candidate Web service and y denoting a particular QoS category, then the normalised value n_{yx} and the score for each candidate x named c_x are:

$$n_{yx} = \begin{cases} \frac{max_y(s_{yx}) - s_{yx}}{max_y(s_{yx}) - min_y(s_{yx})} & \text{for increasing categories} \\ \frac{s_{yx} - min_y(s_{yx})}{max_y(s_{yx}) - min_y(s_{yx})} & \text{for decreasing categories} \end{cases} \qquad c_x = \frac{1}{p} \sum_{y=1}^{p} w_y n_{yx}$$

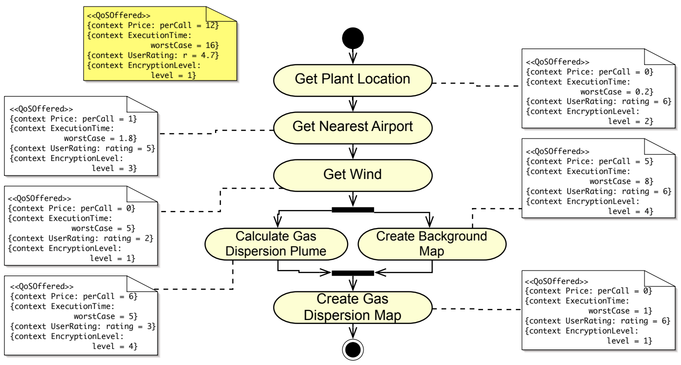
The weight w_y is applied to the categories by the user's preference and the value p represents the number of used QoS categories. The sum of all weights must be equal to 1. The result of this procedure is a score by which the overall QoS of a candidate can be compared with the overall QoS of another.

The outcome of this phase is an assignment of exactly one service for each task in the composition and the resulting aggregated QoS statement that can be applied to the composition. Technically the result can be presented in one file for the QoS offered of the aggregation and one file that identifies the WSDL-operations assigned to identifiers of each task.

5 Step 3: Composition with Selected Services and Offered QoS

The composition model will be updated to show the selected services for each task including the individual offered QoS values of each service and the aggregated offered QoS for the composition as a whole. Figure 7 shows such an updated model for the gas dispersion case with imaginary offered QoS values. The production of this UML model is preferably supported by automatic reverse engineering in a transformation tool that takes the produced files from step 2 as input. The list of selected services will be assigned to its respective tasks and the details identifying the WSDL operation to execute is provided in the model as UML tagged values. To simplify the figure we have not shown these WSDL details, only the name of the provider of the service.

The offered QoS values are taken from the QoS offered documents assumed to accompany each WSDL operation. In UML we propose to specify the offered QoS values in a note attached to each task where the corresponding WSDL operation is chosen. The UML has also an unattached offered QoS note that by our convention indicates that it applies to the composition as a whole. The way of specifying the offered QoS value in the note follows the current draft OMG profile for QoS, while attaching them to the tasks and the activity model as a whole is a new way of applying the OMG profile. The current version of the OMG profile mentions only QoS offered for software components, classes or interfaces. The offered QoS values indicate the promised QoS for the individual services and the new Web service composition.



<<QoSOffered>>
{context Price: perCall = 12}
{context ExecutionTime: worstCase = 16}
{context UserRating: r = 4.7}
{context EncryptionLevel: level = 1}
<<QoSOffered>>
{context Price: perCall = 1}
{context ExecutionTime: worstCase = 1.8}
{context UserRating: rating = 5}
{context EncryptionLevel: level = 3}
<<QoSOffered>>
{context Price: perCall = 0}
{context ExecutionTime: worstCase = 5}
{context UserRating: rating = 2}
{context EncryptionLevel: level = 1}
<<QoSOffered>>
{context Price: perCall = 6}
{context ExecutionTime: worstCase = 5}
{context UserRating: rating = 3}
{context EncryptionLevel: level = 4}
Get Plant Location
Get Nearest Airport
Get Wind
Calculate Gas Dispersion Plume
Create Background Map
Create Gas Dispersion Map
<<QoSOffered>>
{context Price: perCall = 0}
{context ExecutionTime: worstCase = 0.2}
{context UserRating: rating = 6}
{context EncryptionLevel: level = 2}
<<QoSOffered>>
{context Price: perCall = 5}
{context ExecutionTime: worstCase = 8}
{context UserRating: rating = 6}
{context EncryptionLevel: level = 4}
<<QoSOffered>>
{context Price: perCall = 0}
{context ExecutionTime: worstCase = 1}
{context UserRating: rating = 6}
{context EncryptionLevel: level = 1}

Fig. 7. Composition Model with Actual QoS Values

6 Step 4: Exporting the Model

The final Web service composition model with QoS offered is a basis for code generation to WSDL [5], execution specification (e.g. BPEL) [3] and a QoS XML document. Finally the Web service composition is deployed and published as WSDL with accompanying offered QoS values for the service composition as a whole. The WSDL of the gas dispersion case consists of one operation CreateGasDispersionMap() with a reference to the QoS values. The QoS values in the XML file are equivalent to those in the model of Figure 7. The offered price per call is 12 Euro, the worst case execution time is 16 seconds, the user rating is 4.7, and the encryption level is 1.

7 Discussion

The Web service composition methodology is an iterative and continuously evolving process since the available services and their QoS values may change at any time.
New services will come along, other services will have improved or worsened QoS, yet other services will no longer be available. In order to guarantee the offered QoS for the whole composition one needs to negotiate and agree on contracts between the party offering the composite service and each individual service party. When the composite service party is notified of a change to one of its sub-contractor's services, all the steps in the methodology of this paper should be reapplied as a new iteration.

To ensure that new, possibly QoS improving services are utilised, the methodology steps should be reapplied at a regular basis. The first step in the methodology is the only one that requires human intervention. This step can also be fully automated in the future, but this will require a lot of unsolved infrastructure support with respect to semantic properties and standardised use of semantic ontologies. Our methodology assumes that the WSDL operations have references to a document describing the offered QoS values. It is natural to add a language to capture this information to the family of Web service specifications such as WSDL, SOAP and BPEL4WS. It is also natural that this language has an XML notation, as this is true for the existing other Web service specifications. Although there is a need for a new XML specification for capturing QoS information, the concepts have already been well investigated in previous research efforts. The forthcoming OMG profile for QoS will hopefully become a successfully adopted standard for capturing QoS information in UML.

This work is founded by long-term research and practical experience. We therefore propose to define a transformation between this specification and an XML language with the same concepts and terminology, so that mainly the notation is different. Implementation of such a transformation is needed in order to fully automate some of the steps in our methodology. It is outside the scope of this paper to define the transformation.

Pattern-wise Selection vs. Local and Global Selection. A simulation tool has been realised, which benchmarks the pattern selection. Besides a local-greedy and the pattern selection, a third algorithm has also been implemented, which evaluates all possible assignments for all tasks within the composition. This selection, which is using a global perspective, always finds the best assignment considering the whole composition. But it scales exponentially with a growing number of tasks and candidates. The pattern selection can be seen as a specialisation of the global selection. The benefit is that it identifies the potential of QoS optimisation as explained in the example scenario. The algorithm performing a pattern-wise selection scales exponentially only with a growing number of candidates and tasks within a pattern element. Thus it will execute faster than a true global-based approach, if a composition consists of more than one pattern element. The simulation tool has been realised in Java, J2SE 1.4. To ensure the optimal performance, the tool uses primitive data types and their arrays when performing the following steps:

1. Generation of the example structure based on the gas dispersion case.
2. Generation of candidates each with random QoS values for each task.
3. Performing each of the three selection methods each on the same composition with the same set of candidates.

The implementation of the tool, the random generation of the candidates and their QoS involve a large number of issues, which cannot be discussed in detail in this paper due to limited space. However, we would like to mention the following two main points:

- The simulation environment generates random QoS values for a given number of candidate services; however, for each task an optimal QoS is randomly set first and the values for the referring candidates are within zero and 100 percent worse compared to it. This ensures that generated QoS values remain in the same magnitude.
- For each setup with a particular number of candidates the simulation has been repeated for 25 times with different candidates. The results shown are the average over 25 repetitions for one setup.

Figure 8 shows a comparison of the resulting QoS by using the three different selection algorithms. The two upper lines show the ratio, how much better the QoS resulting from a global and pattern selection is compared to a local selection which is set to 1. The lower line shows how the global approach compares to the pattern approach. Figure 9 shows the computation times for the given setup. From these two Figures we can notice that the pattern approach shows that the increase of gained QoS compared to local selection is almost as large as with using the global selection. The computation time increases significantly when using pattern and global selection with an increasing

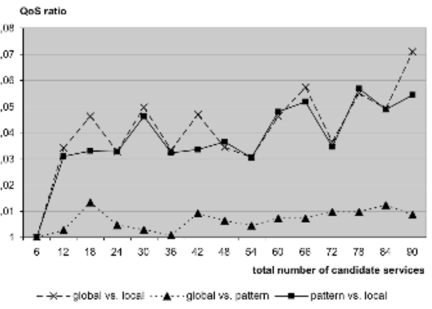


Fig. 8. QoS Ratio for Selection Methods

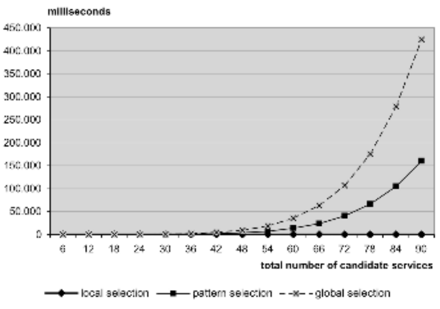


Fig. 9. Duration for Selection Methods

number of candidate services. The local selection does not show a noticeable increase in computation time even with a relatively large number of candidates.

Comparing the absolute results, the global selection takes significantly longer than the pattern: e.g. looking at results for the setup with 60 candidate services, the pattern selection computes in about 13 seconds, while the global takes about 35 seconds. Also, for 90 candidate services the global selection takes about 2.5 times longer.

8 Related Work

The related work of QoS can be sorted in three categories: QoS modelling in UML, QoS Methodologies and QoS-optimisation approaches. In this section each of these fields is presented separately.

QoS Modelling in UML. OMG is in the process of finalising a standard UML profile for modelling QoS [9]. This UML profile defines a way to specify the QoS ontology with QoS characteristics such as we have used in Figure 4. It also has ways to express requirements with QoS requirements, which we have followed in Figure 3. The attachment of these QoS requirements to activities and activity models is an extension introduced in this report. Furthermore the concept of QoS interests is new in this report. Salazar-Zárate and Botella [10] have also proposed a way to use UML to capture similar concepts as OMG. NF-attributes correspond to QoS characteristics, NF-requirements correspond to QoS requirements, and NF-behaviour corresponds to QoS offered. In addition to OMG, they propose to use OCL expressions in a new class compartment to express constraints and inter-dependencies between the QoS model elements. Frølund and Koistinen defines an extension to UML called QML which is an alternative to OMG's forthcoming UML profile. QML also allows for definitions of QoS characteristics, QoS requirements and QoS offered (although with different terminology). QoS requirements can be associated with interfaces that are used by a requester, while QoS offered can be associated with interfaces that are implemented by a provider. OMG's UML profile uses stereotyped notes, while QML uses dashed rectangular boxes to define the QoS requirements and QoS offered. Our approach is more fine-grained by allowing to visually associate QoS requirements and QoS offered with an individual operation represented by a UML activity, while this is handled by textual scoping information in OMG's UML profile and QML. QML does not define a similar construction to our QoS interest.

QoS Methodologies. We have not addressed the reliability problem that advertised QoS values may not be accurate or up to date. Other work has addressed this issue basically in two ways. The first way is explained by Liu et. al [15] who suggest to use active monitoring and consumers feedback to ensure more correct QoS values of the provided services. The other way is to make a signed contract between the consumer and the service provider that ensures a certain QoS level. Contract negotiations and specifications are covered by Frølund and Koistinen in their QML [4]. Wang et al. [14] also deal with contract negotiation and argues that a mediating manager is needed to ensure the promised QoS of many concurrent clients. The execution time of a service may for instance depend on the number of concurrent clients. A mediating manager could handle admission control and resource management in order to ensure that the contracted clients get their entitled QoS, while others are blocked.

QoS-Optimisation Approaches. In our approach we have presented first the aggregation of QoS values and then the selection mechanism for choosing candidate services. This procedure is also found when looking at the evolution of related research. The foundation for the QoS aggregation in compositions can be seen by the work of Cardoso about calculating the QoS for workflows [2]. In his work he has identified different QoS characteristics and defined calculation rules. Because our composition patterns are based on the workflow patterns, we can also support structures like the two OR-split patterns along with the *m-out-of-n-join* constructs. As a consequence the composition patterns can be easier applied to applications, which are derived from the workflow patterns by van der Aalst et al. [12]. Different authors have also discussed in various articles ([8], [1] or [16]), which QoS characteristics might be considered in compositions of Web services. Their contributions has been considered when determining the relevant QoS criteria for our methodology. If needed, the chosen QoS criteria can be easily extended with additional QoS criteria without affecting our methodology itself. For the selection of candidates, Zeng et al. have already identified a local and a global selection [16]. To address the problem that a global selection has unfeasible effort, they have introduced an Integer Programming (IP) solution to compute the optimal assignment of services, which reduces significantly the computation time according to their tests. We plan to test the IP-approach with our simulation tool to deliver a statement on how well it compares to pattern selection.

9 Conclusions and Future Work

We have introduced a UML model-driven methodology for designing and deploying Web service compositions, which uses a QoS-aware selection process for assigning Web service candidates to the tasks of a composition. OMG's forthcoming UML profile for modelling QoS can be used to capture most of the relevant QoS aspects in the different steps of our methodology, although we have proposed to extend it with the ability to specify QoS optimisation interests. Most of the steps in the methodology can be automated by existing or to-be-defined transformations.

We have also discussed different approaches for service selection and compared their characteristics. From our results we can see that different approaches for service selections would fit into different usage scenarios. The global approach optimises by taking all the different execution paths into consideration. It always performs the best QoS-optimised values but takes the longest time to compute. Thus it is not the best choice for run-time selection of services in time-critical applications or with complex compositions involving many tasks and candidate services. The local approach optimises each single task only, without considering the composition structure. It will always have the shortest computation time and has showed the lowest QoS-optimised values. The pattern approach optimises by taking the control flow patterns into consideration and results in almost the same QoS like the global selection while taking a shorter computation time. The summary is that all of the three approaches may be the best choice for a given usage scenario. With more simulation and testing it is possible to establish a run-time choice framework assuming all the three algorithms are available. It is important to consider if the selection process is carried out in run-time or design-time, and especially the users execution time requirement.

In summary we think that a model-driven methodology is the foundation for a tool-supported composition environment and thus along with the support for QoS the necessary contribution for designing compositions in industry applications. For future work we will adapt additional modelling methods besides the UML activity models for designing compositions. A planned enhancement for the selection of Web services is to add the support for semantic description of services as a selection criterion. This provides the opportunity for automated discovery of candidate services as well as creation of the semantic descriptions for the new composition. Another future extension of our approach is to investigate how we can deal with QoS offerings of services that are dynamic with respect to the the number simultaneous clients and dynamic with respect to the input parameter contents.

Acknowledgments

The work of Roy Grønmo is partially funded by the European IST-FP6-004559 project SODIUM (Service Oriented Development In a Unified framework). We would also like to thank Gero Muehl, Gerhard Koehler and Jan Øyvind Aagedal for their valuable input.

References

1. Benatallaah, Dumas, Fauvet, Rahbi, and Sheng. Towards Patterns of Web Services Composition. In *Patterns and Skeletons for Parallel and Distributed Computing*. Springer Verlag, 2002.
2. J. Cardoso. *Quality of Service and Semantic Composition of Workflows*. PhD thesis, Department of Computer Science, University of Georgia, Athens, GA (USA), 2002.
3. Satish Tatte (Editor). Business Process Execution Language for Web Services Version 1.1. Technical report, BEA Systems, IBM Corp., Microsoft Corp., http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/, February 2005.
4. Svend Frølund and Jari Koistinen. Quality of Service Specification in Distributed Object Systems Design. In *Distributed Systems Engineering Journal*, 1998.
5. R. Grønmo, D. Skogan, I. Solheim, and J. Oldevik. Model-Driven Web Service Development. In *International Journal of Web Services Research (JWSR), vol. 1(4)*, October-December 2004.
6. Ching-Lai Hwang and K. Paul Yoon. Multiple Attribute Decision Making: Methods and Applications. In *Lecture Notes in Economics and Mathematical Systems, Vol. 186*. Springer Berlin, 1981.
7. M. C. Jaeger, G. Rojec-Goldman, and G. Muehl. QoS Aggregation for Web Service Composition using Workflow Patterns. In *Proceedings of the 8th International IEEE Enterprise Distributed Object Computing Conference (EDOC)*, Monterey, California, September 2004.
8. D. A. Menasce. Qos issues in web services. In *IEEE Internet Computing, vol. 6(6)*, pages 72–75, November-December 2002.
9. OMG. UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms, August 2003.

10. G. Salazar-Zrate and P. Botella. Use of UML for modeling non-functional aspects. In *Proceedings of the 13th International Conference Software and Systems Engineering and their Applications*, Paris, France, December 2000.
11. W. M. P. v. d. Aalst. Don't go with the flow: Web Services composition standards exposed. In *IEEE Intelligent Systems, vol. 18(1)*, pages 72–76, 2003.
12. W. M. P. v. d. Aalst, A. H. M. t. Hofstede, B. Kiepuszewski, and A. Barros. Workflow Patterns. In *BETA Working Paper Series*, WP 47 2000.
13. I. Walsh, P. McCarthy, and J. Shields. e-Emergency Pilot Demonstrator, e-blana, Deliverable IST-2001-37724 ACE-GIS D1.2b, November 2003.
14. G. Wang, A. Chen, C. Wang, C. Fung, and S. Uczekaj. Integrated Quality of Service (QoS) Management in Service-Oriented Enterprise Architectures. In *Proceedings of the 8th International IEEE Enterprise Distributed Object Computing Conference (EDOC)*, Monterey, California, September 2004.
15. A. H. H. Ngu Y. Liu and L. Zeng. QoS Computation and Policing in Dynamic Web Service Selection. In *Proceedings of the Thirteenth International World Wide Web Conference*, New York, USA, May 2004.
16. L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalgnanam, and H. Chang. QoS-Aware Middleware for Web Services Composition. In *IEEE Transactions on Software Engineering, vol. 30(5)*, pages 311–327, May 2004.

How to Implement Software Connectors? A Reusable, Abstract and Adaptable Connector

Selma Matougui and Antoine Beugnard

ENST-Bretagne, Technopôle Brest-Iroise,
CS 83 818, 29 238 Brest, France
{selma.matougui, antoine.beugnard}@enst-bretagne.fr

Abstract. In recent software developments, applications are made up of a collection of reusable software entities (components) and mechanisms that permit their interaction (connectors). These latter mechanisms have many forms. On the one hand, industrial approaches use simple connectors that are mainly point-to-point connections. On the other hand, academic approaches, like Architecture Description Languages (ADL), recognize complex connectors as first class design entities. However, these concepts are restricted to the architectural level since they have almost no implementation. The current application developments use simple connectors, and high level specifications are under exploited.

In this article, we propose a means to fill the gap between connector specification and implementation. For a better reuse of design effort, and to avoid using only simple connectors when realizing applications, we propose to define connectors as complex communication and coordination abstractions and to implement them as a family of generators. We illustrate the development and use of such generators through a full example.

1 Introduction

As software systems become more complex, applications are being constructed from reusable, interconnected software components and connectors. For a long time, components have been considered as the fundamental building blocks of software systems. Although there is still no universal definition of components, there is a kind of consensus about what a component is: it offers and requires services, and performs some computation. The interconnection mechanisms, or connectors, play a deterministic role in establishing the global system properties but their level of understandability is still far from the level reached by components, as argued in [1]. The goal of this paper is to propose a better way to implement abstract connectors.

In the remainder of this article, section 2 analyses the problem and depicts what is wrong with current understanding of interconnection mechanisms. Next, section 3 details our proposal where we define what a connector is and explain its life cycle. Then, section 4 illustrates the way to implement and use such connectors with a load balancing connector. After that, we make relationships with other work in section 5, and finally we conclude in section 6.

2 Motivation

Industrial component approaches like CCM [2] and EJB [3], do not refer the notion of connector. Interaction mechanisms are low level. They refer to a limited set of simple communication abstractions among which we can cite the synchronous communication embodied as Remote Procedure Calls (RPC). These interaction mechanisms are the most used in current application developments and several implementations exist like CORBA, RMI, and SOAP. Their main functionality is to transport data and synchronize the interacting components. They are not able to make any decisions for coordinating components. The lack of abstraction implies realizing a complex communication with a sophisticated combination of simple interactions included in the functional components. Developers are constrained to construct and construct again the complex communications over these platforms and applications, anytime they need to use them, in spite of the fact they reuse them. The effort of realizing a complex communication is not well exploited. For a better quality of software and separation of responsibilities, however, we would like to be able to include in connectors more communication and coordination functionalities than only those of transporting data [4, 5].

In academic approaches like Architecture Description Languages (ADL [6]), the interaction mechanisms have been recognized as first-class design entities in software architectures and are defined as connectors [7]. We identify 3 kinds of ADL that express connectors differently. The first type defines a set of connectors implemented as simple ones, like Unicon [8]. They almost meet the interaction mechanisms of industrial approaches. The second type defines connectors that model complex interactions between components like Wright [9]. Unfortunately, these connectors are reduced to specification descriptions and have no implementation. These specifications are under-utilized and current developments always refer to simple connectors. The third type defines complex connectors, like Rapide [10], but provide a fixed set of connectors types designed as components that have explicit interfaces. Hence, to interact with these connectors, the functional components explicitly call the services offered by the connector. If the connector were changed for one reason or another, the component would use the new interfaces of the new connector. Thus, we often need to use adaptors between the components and the connectors if their interfaces do not meet. These various definitions of connectors indicate how poorly they are understood. We are far from a consensus about what a connector is.

Hence, there is not a standard way to represent connectors. When comparing industrial and academic approaches we notice that industry uses simple connectors because they are implemented and available. Unfortunately, complex communications are always constructed (and constructed again) over existing platforms and are mixed with components. This makes applications hard to maintain and limits component and connector reuse and evolution. In academic approaches, some ADL define complex communication abstractions, but they do not offer a standard way to design and implement connectors. They either specify complex connectors without offering means to implement them, specify and implement simple ones [6], or implement them as components, but this last solution lacks adaptability. Almost all languages claim to make interactions as connectors but without meaning the same thing all the time.

Regarding this discussion, we support the need to model and specify complex interactions as autonomous entities in software architecture just as many other works have done. Nevertheless, the question to ask is how to implement these connectors in order to offer and preserve the abstraction and to ensure an automated adaptation. Existing approaches offer ad hoc and predefined answers. In this article, we propose a global vision of what is a connector and propose to implement it as a family of generators in order to ensure both abstraction (of connectors) and adaptation (to components). In this way, we offer a better and easy configuration or reconfiguration of applications, component and connector reuse, and a better exploitation of design effort.

3 Connectors: Definition and Life Cycle

A connector is a reification of an interaction, a communication or a coordination system. At an abstract level, it exists to serve the communication and coordination needs of unspecified interacting components. Later in the life cycle, it describes all the interactions between special components by relying on the own interface specifications of these components. It also offers application-independent interaction mechanisms like security and reliability, for instance. Three points distinguish a connector from a component:

– It is an autonomous entity at design level but neither deployable nor compilable (it does not exist in conventional programming languages). The connector is an architectural entity. It must be connected in order to be deployed.
– It does not specify offered or required services but patterns of services. Hence, the interfaces are abstract and generic. They become more concrete later in the life cycle, on assembly with the components. This is done by adopting the interfaces type of the interacting components.
– In order to distinguish services grouped in interfaces of component ports, the word *plug* is introduced to name the description of connector ends. A plug is typed with a name and can have a multiplicity.

We distinguish four stages in the life cycle of a complex connector, summarized in table 1 and illustrated in figures 1 and 2. The connector evolves in time and, for each stage, we give it a different name and a particular graphical representation with abstract parts (dotted lines) and concrete parts (solid lines). Connectors are represented by an ellipse, in order to differentiate them from components, and plugs are represented by circles[1] around the ellipse.

[1] In opposition with components representation as boxes-and-lines [11], where components are boxes and their defined interfaces are squares. We illustrate our complex connectors by an ellipse in order to differentiate them from the simple ones that are represented by lines.

Table 1. Vocabulary by level of abstraction and refinement

Level	Architecture (abstract)	Implementation (concrete)
Off-the-shelf	*Connector*	*Generators*
Assembled	*Connection*	*Binding components*

In its most abstract state, figure 1 (a), this entity is called a *connector*. The sole concrete parts that are defined are the properties that have to be ensured and the number of interacting components able to be connected (number of different plugs). They are in fact the sole invariants that exist during the whole life cycle. The abstract parts to be specified later are the protocols of the underlying platforms on which the connector will be implemented, and the interfaces of the interacting components (both in gray in figure 1 (a)).

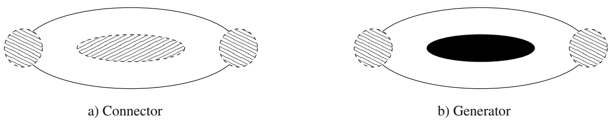


Fig. 1. The connector alone : (a) abstract, (b) implemented

An isolated connector is concretized as a family of off-the-shelf *generators*. These generators are in charge of the implementation of the connector abstraction over different communication protocols or platforms. The plugs are still abstract, the interacting components are not specified yet. When activating the generators, the plugs are destined to be transformed into proxies towards which the effective communication will take place. Figure 1 (b) shows the transformation of the internal part of the connector into a concrete entity.

At an abstract level, it is possible to use the connector in a software architecture. The connector is linked to other components, and form the *connection*. At this stage, the generic interfaces of the connector (plugs) become concrete and adopt the applicative interfaces (APIs) of the interacting components. In figure 2 (a), the plugs (circles) are concrete because their type is specified, they form the connection interfaces. Hence, we connect the components without worrying about the underlying platform.

The most concrete form of the connector is represented in figure 2 (b). Both the protocol and the actual interfaces are known. The generator, that was de-

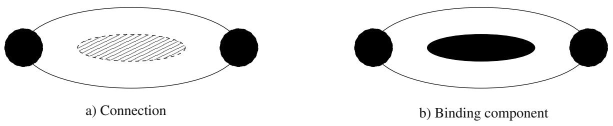


Fig. 2. The connector assembled : (a) abstract, (b) implemented

veloped for the chosen protocol, can use these actual connection interfaces. It generates the proxies that represent the realization of a communication or coordination property for the connected components' requirements (APIs) over the underlying system. We refer to this entity as the *binding component*[2].

From an application developer's point of view, the software architecture definition involves selecting components and connectors off-the-shelf, assembling (*i.e.* establishing connections), and generating the application onto the target system with the appropriate generators.

From a connector developer's point of view, defining a connector involves the specification of the abstract communication properties (load-balancing, for instance), and the development of the generators that use the interface definition of connected entities (IDL, for instance) over a target system (TCP/IP, for instance). As we can see, a family of generators could be developed for a single abstract connector.

An example of an existing generator, as defined in this section, is the CORBA generator as an instance of the RPC connector. The properties it ensures are distribution, language interoperability, and a RPC semantics. It can rely on several communication protocols like TCP/IP or other low level protocols. The proxies it generates are the stub and the skeleton.

Hence, we use this definition and life cycle of what a connector is to implement or realize complex communication abstractions. Indeed, to ensure properties like authentification, data compression, consensus, cryptography, or load balancing, current developments mix them with the application code. By dedicating one connector to realizing such non functional properties thanks to the generation process, we offer the possibility to provide separation of concerns transparently to the application and independently of the platforms.

In the following we will demonstrate the whole process of a connector specification, implementation and use through a load-balancing connector.

4 Load Balancing Connector

In this section we illustrate, throughout an example, the whole connector life cycle and the benefits of using our approach. We begin by presenting briefly the main features of the chosen complex interaction property that the connector holds: load balancing. Then, we detail the description of the connector and the different steps it goes through. Finally, we show an implementation of the load balancing connector. Actually, the aim of this section is to show the feasibility of our proposal and the advantages of such a software engineering approach.

4.1 Load Balancing Connector Features

Some applications, like e-commerce systems and online stock trading systems, concurrently service many clients transmitting a large, often bursty, number of

[2] It is a refinement of the *binding object* defined in Open Distributed Processing (ODP) [12].

requests. In order to improve the overall system responsiveness, an effective way to deal with this great demand is to increase the number of servers — replicas offering the same service — and to apply a load balancing technique. Load balancing mechanisms distribute client workload equitably among back-end servers by indicating which server will execute each client's request. These mechanisms may follow either a *non-adaptive* or an *adaptive* policy. Non-adaptive policies do not take the servers' workload into account. The round robin is such a policy. Adaptive policies, however, take the servers' workload into consideration. The policy of the least loaded server is an example. Many load metrics can be used; the CPU's idle cycles or the number of requests, for instance. In the following example, we assume a distributed application where a client needs a defined service offered by a server. As a client needs to send a huge number of requests, these requests would be balanced among n replicas of servers following a special load balancing policy.

Performing load balancing at an OS [13] or a network [14] level does not allow to use application metrics or control the load balancing policy without modifying the application itself. Hence, we propose to achieve load balancing mechanisms at a high level (eg. application level) but *transparently to the clients' and the servers' codes*. We reify connection abstractions as an independent entity. For instance, we consider load balancing as a complex interaction that involves many participants and is achieved with different protocols and policies. This abstraction is the *connector*. The property it ensures is load balancing and its number of plugs is two. Indeed, the connector distributes incoming requests of one kind of component (clients) to one kind of another component (servers). These plugs are of multiplicity n because the connector is able to make interacting several (same) clients with several (same) servers. The service offered and required is the same both side, but is unknown *a priori*. This service is not known when the load balancing connector is developed.

4.2 Load Balancing Connector Life Cycle Description

The load balancing connector is designed and implemented as set of code *generators*. A generator is in charge of producing the code that allows to ensure the load balancing mechanisms on a target platform or system. The different generators differ according to different criteria. They can differ in relation to target platform or policy differences. For example, if we are in a distributed environment, the generators could be built over different available platforms that allow remote communications like CORBA or RMI. Differentiating the generators from a policy point of view gives birth to as many generator implementations as existing policies. By combining these criteria, we obtain several generator variants for one connector. Hence, we can have a load balancing generator with the round robin policy on the CORBA platform or on the RMI platform. These off-the-shelf generators are intended for ensuring load balancing mechanisms on different platforms for initially unspecified components. So, at this stage, the plugs are still generic, they wait to be associated with components' interface descriptions that are used by generators for the effective code generation.

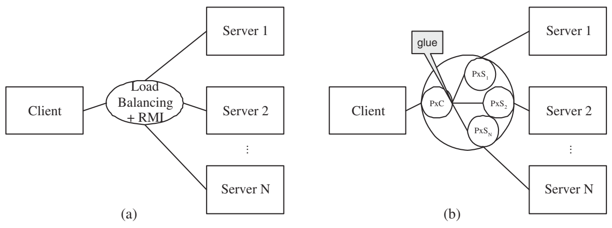


Fig. 3. Complex communication with load balancing connector. (a) connection, (b) binding component

The load balancing connector is assembled with the components of the application: the client and the servers. Figure 3 (a) represents the result of this assembly: *the connection*. Henceforth, connector plugs know who is interacting, so they take form and adopt the interacting components interfaces. The plug at client's side will embody the services offered by the servers. The client deals with the plug as if there was only *one* server (as a *shared* proxy). The plug at the server's side will embody the services needed from *one* client. The server is not concerned by managing the different clients that require the service. This process of transforming plugs into complementary interfaces of the interacting components maintains the original functionality of the application; that is the client needs a service from the server. Therefore, the load balancing abstraction is ensured without taking into consideration what is offered or not by a target platform. At this stage, the connection interfaces are specified. They can be provided for one or several generators that are responsible for resolving platform details. These connection interfaces differ from one application to another, as the component services they have adopted differ.

Once the connection established, the resulting connection interfaces are given as parameters to a chosen off-the-shelf generator[3] that activates the generation process of *the binding component*. Both the plugs and the underlying platform are now well known. The generation process can transform the connection interfaces, mapped on the generator plugs, into dedicated proxies. Figure 3 (b) highlights these generated proxies that represent the realization of the load balancing property for the client and the server interfaces over the chosen underlying platform, for example RMI. If the selected generator is built over RMI, the generated proxies include both load balancing and distribution mechanisms. Therefore, the proxy (PxC), at client side, is in charge of converting data and deciding which server will serve the client's requests according to the policy. The proxies (PxS_i), at server side, are in charge of converting data, managing client requests, and reporting server workload in the case of an adaptive policy. For one connection we obtain different binding components by applying different generators.

The load balancing connector represents a new communication abstraction. It is realized as a family of generators. Each generator is designed and implemented,

[3] The generator choice can be done to meet the components' platform for example.

once for all, for generating complex and repetitive details over a platform. The load balancing connector is assembled each time with components that need to balance load. These components may use different services. As connector's plugs are generic, they fit to any interacting component interfaces. Once the generators selected and the plugs associated with its missing interfaces, the binding component can be generated.

This approach ensures both the separation of concerns and the separation of the abstraction and the implementation. The communication and coordination concerns are separated from the functional concerns of the applications. The client and the servers have no extra code; they are entirely dedicated to their functional properties. Hence, this approach offers the means to discharge the application from cumbersome code. Applications become more understandable, easier to maintain and to make evolve. The "know-how" of the abstraction implementation is built up into generators that can be reused any time a new connection takes place. So, it is clearer to application developers. They no longer need to worry about the implementation of connectors. This allows them to change or add a policy without affecting the application, and even use the client without any load balancing connector. They are not concerned with planning the use of load balancing when designing components.

This plug/connector approach is different from the classical role/connector approach. In the latter, the client has to play the role defined by the connector. This implies, most of the time, that the client is adapted to the interfaces of the connection. However, in the plug/connector approach, the client does not need to add extra code to be adapted to the needs of the interaction, it is the plug that adopts the clients needs. The client required service is provided to the client improved by the load balancing and distribution details, that are superfluous for the application.

4.3 Load Balancing Connector Implementation

We have specified the load balancing connector and realized the associated generators. In order to evaluate the whole connector life cycle, we have implemented the load balancing connector in the open source project Jonathan [15]. This framework basically offers a CORBA and RMI-like generators for the RPC connector. We have developed generators to realize the load balancing connector with 2 variant strategies: round-robin and least loaded sever over Jeremie, the RMI personality of Jonathan. In the following, we briefly explain the implementations of the load balancing generators and their use. All the details can be found in[16].

The first implementation is for the round robin policy. For this non-adaptive strategy, monitoring the servers' workload is not necessary. We have realized the generator as an extension of the JRMICompiler, the standard RMI generator of Jeremie. This new generator, called JRMICompilerLB, has been implemented to ensure both load balancing and distributed properties. It generates the classical proxies that hold the code for converting data to be sent through the network. In addition, JRMICompilerLB augments these proxies with the code of the round robin policy. The standard JRMIRegistry of Jeremie, that holds the different remote server references, has also been modified to JRMIRegistryLB.

In our client/server application, the client uses the following interface which is implemented by the N servers put at the client's disposal to balance load.

```
public interface ApplicationInterface{

    public int operation(); // the server's service

}
```

At assembly time, when an architect assembles the off-the-shelf components and the load balancing connector, the abstract plugs of the load balancing connector adopt this ApplicationInterface in order to apply the semantics of the communication to it. Hence the connection interfaces become concrete. Since these connection interfaces are known, they are provided as parameters to the generator in order to generate the appropriate proxies. The following command line is used:

```
> JRMICompilerLB ApplicationInterface
```

Once the proxies generated, they are deployed with their attached components. So when the client calls the server's operations, the proxy intercepts them locally. The proxy always performs its initial functionality of converting data, but in addition, has the responsibility for coordinating the back-end servers by applying the round robin policy. It forwards every request to the next replica in the group of servers registered in the JRMIRegistryLB. This is done by replacing the current remote reference of the (PxC) proxy (see figure 3) at each request by the delivered reference from the JRMIRegistry. Hence, at run time, the client sends a request locally to the sole proxy in its possession; the proxy embodies the server's service but does not perform it; it forwards the request to the selected server according to the appropriate policy.

In the second implemented load balancing connector, the generator generates code to ensure the the least loaded server policy. In this adaptive policy, clients' requests are sent to the servers according to their load. The policy has been implemented in a specific publish/subscribe component. The generated proxies (PxS_i), at the server side, publish the load they are in charge of monitoring. The servers are not aware of this load monitoring. The publish/subscribe component chooses the least loaded server that will serve the request. All subscribing clients' proxies are informed with this new reference and use it when a service request is made. We choose to implement this generator as an extension of JRMICompilerLB generator with the option *-lbAdaptive*. The publish/subscribe component, the proxies and the JRMIRegistryLB, therefore, make up the binding component.

We have evaluated performances of the new generator with the round robin policy over a network of 32 machines with 1 to 8 servers and 1 to 22 clients[4].

[4] One machine was reserved for a registry and another one for controlling the benchmark.

The results obtained from a latency test showed no overhead due to reference switchings and demonstrated a good scalability. For example, the time needed to serve the 22 clients with one server using the load balancing connector was 1.85 ms, and the time needed to serve the same 22 clients with 8 servers was 0.75 ms. All the results can be consulted in [16].

It is worth noticing that Jonathan's previous generator can still be used to implement point-to-point RMI connections. So, introducing load balancing, or changing the load balancing strategy, requires a simple proxy regeneration from the same interface description. This approach helps to easily reconfigure the application and change the policies only by changing the connector and regenerating the proxies.

5 Related Work

There are very few works related to the generation of connector implementation.

In [17], Garlan and Spitznagel define operators to create new complex connectors by composing simple ones. As we have seen in section 4.3, in our approach we can build complex connectors from simple ones (extension of RMI). We can also build them from a combination of a component with a generator (using a publish subscribe component [16]). Their approach is simple to implement, because they reuse existing connectors without any modification. However, this involves several intermediate steps because of the combination of several connectors. Thanks to the generation process, our approach enables us to change some interaction mechanisms that are not appropriate to the interaction requirement. Moreover, Garlan and Spitznagel's implementation passes directly from the specification to the binding component step through the connection—according to our life cycle. Their approach does not have off-the-shelf generators for different technologies, so they have to make the same transformations whenever a composition has to be created over a different platform. We argue that the two approaches are complementary. Owing to the fact that our approach is more complex to realize, we believe it to be more efficient, since the transformations they propose could be automated with the development of generators.

The work on ArchJava [18] also uses a kind of generator for connectors. It is more focussed on implementing a connector on a Java/ArchJava platform and uses dedicated classes to implement the connectors. These specific classes could be seen as our generators. Our work is an attempt to offer a better conceptualization frame in order to generalize the use and implementation of connectors.

6 Conclusion

Software architecture is playing a more and more important role in software engineering. It is well recognized that architecture is of crucial importance in all nonfunctional properties of software applications. We have shown in this paper that, despite their importance, some key concepts such as connectors are still illdefined. We have proposed a new vocabulary in order to help software architects and designers to better conceptualize architectures.

Connectors have adaptable implicit interfaces that provide communication properties transparently to the application. For instance, changing a consensus connector for a load-balancing one requires a new proxy generation that does not require any modification at the application level but does have a strong impact on the nonfunctional properties of the whole system. From a software development life cycle, we have introduced a clear vocabulary in order to distinguish all the connector stages. A *connector* is an abstract software object whose intention is to implement a communication property. This abstraction may have several implementations, called *generators*, depending on design choices and on the target platform. When abstractly used in an architecture in order to link components, we call it a *connection*. Finally, the deployed and executable stage is called a *binding component*.

We believe that this conceptualization suggests to developers to design new kinds of generators, as we have illustrated with a load balancing connector, instead of always reusing the same "old-good-one". We believe that accumulating connector development experience through developing generators is the right way to use and reuse connectors. It ensures abstraction through the generation process, and adaptability thanks to plugs.

Finally, we believe there are two ways of producing reusable software parts: usual off-the-shelf components, that can be *directly* integrated in a software architecture, and off-the-shelf generators, that are used *indirectly* to generate the adaptable binding components that link application components together.

References

1. Mehta, N.R., Medvidovic, N., Phadke, S.: Towards a taxonomy of software connectors. In: the 22nd International Conference on Software Engineering (ICSE 2000). (2000) 178–187
2. OMG: CORBA Component Model RFP (1997) `http://www.omg.org/docs/orbos/ 97-05-22.pdf`.
3. Inc, S.M.: (Enterprise java beans technology) `http://java.sun.com/products/ejb/`.
4. Shaw, M.: Procedure calls are the assembly language of software interconnection: Connectors deserve first-class status. Technical Report CS-94-107, Carnegie Mellon University, School of Computer Science (1994)
5. Ducasse, S., Richner, T.: Executable connectors: Towards reusable design elements. In Verlag, S., ed.: ESEC/FSE'97 (European Software Engineering Conference). Volume 1301 of LNCS (Lectures Notes in Computer Science). (1997) 483 – 500
6. Medvidovic, N., Taylor, R.N.: A classification and comparison framework for software architecture description languges. In: IEEE Transaction on Software Engineering. (2000) 70–93
7. Show, M.: Procedure calls are the assembly language of system interconnection: Connectors deserve first-class status. In: Workshop on Studies of Software Design. (1993)

8. Shaw, M., DeLine, R., Zelesnik, G.: Abstractions and implementations for architectural connections. In: Third Int'l Conf. Configurable Distributed Systems. (1996)
9. Allen, R., Garlan, D.: A formal basis for architectural connection. ACM Trans. Software Eng. and Methodology **6** (1997) 213–249
10. Luckham, D., Kenney, J., Augustin, L., Vera, J., Bryan, D., Mann, W.: Specification and analysis of system architecture using rapide. IEEE Trans. Software Eng. **21** (1995) 336–355
11. Abowd, G.D., Allen, R., Garlan, D.: Using style to understand descriptions of software architectures. ACM Software Engineering Notes **18** (1993) 9–20
12. Putman, J.: Architecting with RM-ODP. Prentice Hall (2001)
13. Coulouris, G., Dollimore, J., Kindberg, T.: Distributed systems: Concepts and design (2001)
14. Inc., C.S.: High availability web services (2000) `http://www.cisco.com/warp/public/cc/so/neso/ibso/ibm/s390/mnibm_wp.htm`.
15. Middleware, O.O.S.: (Jonathan: an Open Distributed Objects Platform) `http://jonathan.objectweb.org/`.
16. Sanchez, F.J.I., Matougui, S., Beugnard, A.: Conception et implémentation de connecteurs logiciels : Une expérimentation pour le calcul de performance sur une application de répartition de charge. Technical report, ENST-Bretagne, Brest, France (2004)
17. Spitznagel, B., Garlan, D.: A compositional approach for constructing connectors. In: The Working IEEE/IFIP Conference on Software Architecture (WICSA'01). (2001) 148–157
18. Aldrich, J., Sazawal, V., Chambers, C., Notkin, D.: Language support for connector abstractions. In: European Conference on Object-Oriented Programming (ECOOP '03), Darmstadt, Germany (2003)

Designing Self-adaptive Multimedia Applications Through Hierarchical Reconfiguration

Oussama Layaida and Daniel Hagimont

SARDES Project, INRIA Rhône-Alpes
First.last@inria.fr

Abstract. Distributed multimedia applications are very sensitive to resource variations. An attractive way for dealing with dynamic resource variations consists in making applications adaptive, and even self-adaptive. The objective is to grant applications the ability to observe themselves and their environment, to detect significant changes and to adjust their behavior accordingly. This issue has been the subject of several works; however the proposed solutions lack flexibility and a high-level support that eases the development of adaptive applications. This paper presents PLASMA, a component-based framework for building multimedia applications. PLASMA relies on a hierarchical composition and reconfiguration model which provides the expected support. The experimental evaluation shows that adaptation can be achieved with a very low overhead, while significantly improving QoS of multimedia applications as well as resource usage on mobile equipments.

1 Introduction

Recent advances in mobile equipments and wireless networking have led to the emergence of a wide range of peripherals such as, Personal Digital Assistant (PDA), hand-held computers, Smart Phones, eBooks, etc. The Internet infrastructure became, like never before, heterogeneous and dynamic. System and network resources such as network bandwidth, CPU load or battery life time are characterized by unpredictable variations making difficult to guarantee the correct execution of multimedia applications.

The most attractive approach to deal with this issue consists in making these applications self-adaptive, that is, grant them the ability to observe themselves and their environment, to detect significant changes and adapt their own behavior in QoS-specific ways. A well recognized approach to achieve it is the use of component-based technologies [1]. The common idea consists in implementing multimedia-related functions in separate components. Various multimedia services can then be built by selecting and assembling the appropriate ones within the same application. Likewise, adaptation is achieved by means of high-level component reconfigurations such as: adjusting component properties, stopping/starting a subset of components, removing/inserting components or modifying their assembly. Complex operations can be made-up of combination of those basic operations, performed in an appropriate order.

This article describes PLASMA, a component-based framework for the development of self-adaptive multimedia applications. PLASMA relies on an advanced component model [2], whose main features are: recursive composition and hierarchical reconfiguration management. This allows to model reconfiguration in a generic way thus addressing arbitrary applications and adaptation policies. The remainder of the article is organized as follows: Next section presents a classification of related works. Section 3 introduces our design choices and the PLASMA architecture. Section 4 presents an application use case and a performance evaluation of PLASMA. Finally, section 5 concludes this paper and presents perspectives.

2 Related Work

As previously mentioned, adaptivity is tightly linked with component-based technologies. Work around multimedia applications has led to several component-based frameworks such as DirectShow (Microsoft) [6] JMF (Sun) [13] and PREMO (ISO) [7], easing the development of multimedia applications. Following this vein, the advantages of component-based technologies have motivated several research works in order to bring adaptivity to multimedia applications [3, 15, 14] but very few of them considering run-time reconfiguration. This feature has been addressed with different approaches:

- ***Static Reconfiguration Policies***: A first approach to reconfiguration uses static reconfiguration policies to deal with specific resource variations. VIC [17] is a well-known example of such applications. Although it is not component-based, it uses the RTCP [10] feedback mechanism and a loss-based congestion control algorithm [11] that adapts media streams to the available bandwidth. Reconfiguration operations consist in tuning key encoding properties (quality factor, frame rate, etc.) in order to adjust the transmission rate appropriately. Nevertheless, the use of static reconfigurations is too restrictive as they have to be anticipated at development-time.
- ***Component-Based Frameworks with Reconfiguration Capabilities***: Some component-based frameworks grant application developers with enhanced reconfiguration capabilities. The Toolkit for Open Adaptive Streaming Technology (TOAST) [8] investigates the use of open implementation and reflection to ease the development of adaptation strategies. However, it remains to the responsibility of the application developer to deal with resource and application monitoring, reconfiguration decisions and their implementation, which is a pretty heavy task.
- ***Component-Embedded Reconfiguration Policies***: To fill this gap, some works propose to integrate reconfiguration features in the functional components themselves. In Microsoft DirectShow [6] for example, processing components (called filters) exchange *in-stream QoS messages* traveling in the opposite direction of the data flow. Using this mechanism a component may indicate to its predecessors that data is being produced too rapidly (*a flood*) or too slowly (*a famine*), which decrease (or increase) their data processing rate in response. Such mechanism can be easily extended to support a larger scope of QoS control, as

proposed in [5, 19]. However, the limitation of this approach is that reconfigurations only occur in the scope of each component rather than in that of the application. Although it is possible to change the behavior of a given component, it is not possible to perform a component replacement.

- ***Separate Reconfiguration Manager***: By opposition to the previous approach, some works have proposed that all reconfiguration features be integrated in separate managers. Instead of sending QoS messages through components, they are delivered to a reconfiguration manager, which performs reconfiguration operations. This approach is applied in the *DaCapo*++ [12] communication framework, in the *2KQ* framework [16] for resource management and in CANS (Composable Adaptive Network Services) [9] and APC (Automatic Path Creation Service) [18] to design adaptive media transcoding gateways. The main limitation of this approach is that such a manager is tightly-coupled with the targeted application, and especially its architecture. Any modification of the application architecture requires an update of the manager's implementation[1].

3 Component Architecture

The previous section has shown the limitation of previous approaches in addressing requirements mentioned before. This limitation resides in the use of flat component models. Indeed, the integration of reconfiguration at the component level limits reconfiguration capabilities. At the other hand, making it separated from the application requires strong assumptions on its structure. Hence, a new component model becomes necessary with the goal of addressing a large scope, or better, all possible adaptation algorithms within arbitrary component assemblies. This section describes how this requirement is addressed in PLASMA.

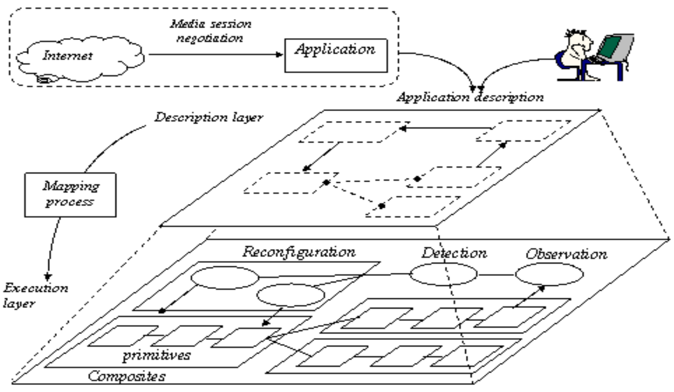


Fig. 1. Overall architecture design

[1] Or it should be explicitely managed in the manager's implementation.

3.1 Design Choices

As depicted in Figure 1, the architecture of PLASMA is composed of: a *description layer* providing tools for the description of applications and their adaptation policies, and an *execution layer* for the composition and the execution of applications.

- *Using a Dynamic ADL*: An application is described using a dynamic ADL (Architecture Description Language). It offers constructs for the description of the application architecture as well as its adaptation policy in terms of observations, detection of relevant changes and processing of reconfiguration operations. All these operations are modeled within the language in a generic way so that it is possible to describe every possible application and its adaptation policies. Descriptions can be written by developers as configuration files or automatically generated by applications requesting specific multimedia services. The framework provides tools to translate a given description into the appropriate component assembly in the execution level. A detailed example of the ADL language is presented in section 4.
- *Recursive Composition*: The construction of applications is facilitated by a recursive component model. An application is divided into several parts, called composite components, which are in turn defined with sub-components. The model allows arbitrary number of levels; the lowest includes primitive components encapsulating functional code. This model is based on the Fractal composition framework [2].
- *Hierarchical Reconfiguration Management*: Reconfiguration policies can be defined at any level of the component hierarchy. Each component has its own reconfiguration manager responsible for reconfiguring its content. In a primitive component, this manager acts on its functional code, for instance by modifying parameters. In a composite component, it applies reconfiguration on its sub-components, transparently to higher levels. This model allows dividing reconfiguration responsibilities into several hierarchical managers, each dealing with a specific part of the application.
- *Composable Adaptation Policies*: Adaptation operations require mainly: the observation of application and resource states, the detection of relevant changes and activation of reconfiguration operations. These functions are implemented in PLASMA as separate and reusable components that can be used to compose every possible adaptation policy.

3.2 Component Architecture

The PLASMA component architecture encompasses three kinds of components: *Media*, *Monitoring* and *Reconfiguration* components. The following subsections detail the role of each of them.

3.2.1 Media Components

Media components represent the computational units used for the composition of the multimedia applications. An application is decoupled into three hierarchical levels, each providing a specific functionality:

- *Media Primitive* (MP) components are the lowest-level processing entities. They implement basic multimedia-related functions such as MPEG decoding, H.261 encoding, UDP transmission, etc.
- *Media Composite* (MC) components are composite components which represent higher-level functions such as: Decoding, Network Transmission, etc. Each media composite deals with a group of MPs and is responsible for their creation, configuration and reconfiguration, transparently to its outside. In Figure 2, an *InputStream* composite is composed of three Media Primitives in the case of an RTP input stream: an RTP receiver, a *Demultiplexer* (Demux) to separate multiple streams and a Depacketizer to reconstitute media data. On the other hand, an HTTP input stream requires one primitive component: *HTTP-Receiver*.
- The *Media-Session* (MS) component is a composite which encapsulates MCs. The Media-Session represents an application configuration and exposes all control features that can be made upon it (i.e. VCR-like operations: start, pause, stop, forward, etc.).

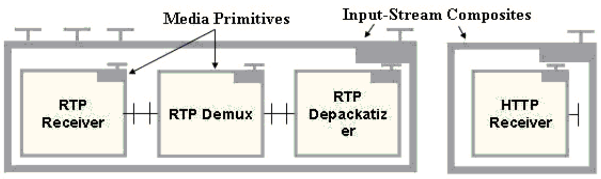


Fig. 2. Examples of Components and Bindings

The advantage of this hierarchy is that it groups primitive components implementing a similar multimedia function under the control of the same composite. This allows the integration of common configuration and reconfiguration operations in the enclosing component independently from the application structure. The combination of various composites constitutes all possible configuration and reconfiguration operations that can be operated on the application. Notice that other levels can be easily added to the component architecture in order to define new composition semantics.

The composition of an application is performed by binding the different components in a flow graph. Both Media composites and Media Primitives participate in this process. Each media primitive exposes one or more stream interfaces used to receive/deliver data from/to other components. A stream interface is typed by the media type of the produced/consumed stream, which is expressed with media-related properties such as MIME type, encoding format and data-specific properties (resolution, colors, etc.). Thus, the success of a binding between two primitive components is governed by the media-type compatibility between the bound interfaces. That is, a binding will fail if there are mismatches between media types of two components. PLASMA provides a binding algorithm which avoids such failures using two kinds of bindings:

- Primitive bindings bind two components agreeing the same media type. This means that media data is streamed directly from input streams to output streams by using method calls between stream interfaces.
- Composite bindings are special composite components which mediate between components handling different media types. Their role is mainly to overcome media type mismatches between MPs. These bindings are made-up of a set of MPs implementing fine-grain media conversions. Figure 3 shows an example of a composite binding between a *Decoder* and an *Encoder* composite. The Decoder provides video data in YUV, while the Encoder accepts only RGB. Moreover, as this later uses H.261 encoding, it only accepts video data in specific resolutions such as QCIF (176*144). The composite binding creates two primitive sub-components: a Resizer to transform the video resolution into QCIF and a YUV-2-RGB to convert data format from YUV to RGB.

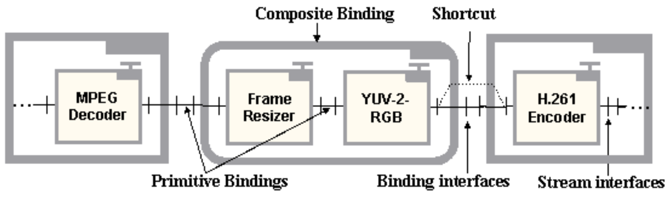


Fig. 3. Examples of component bindings

3.2.2 Probes

Probes define observation points in charge of gathering performance information on both application and system resources. The information is not processed by probes, but only made available to other components wishing access to it. However, they may produce data at different scales and units and thus apply conversions. PLASMA provides two kinds of Probes:

1. *QoS Probes*: Some components are expected to maintain information reflecting QoS values. For example, an RTP Sender component continuously measures packet loss rate, transmission rate, etc. QoS Probes interact with those components to collect QoS information and make it available for other components.
2. *Resource Probes:* They act as separate monitors for gathering resource states such as CPU load, memory consumption, remaining battery life-time, etc.

3.2.3 Sensors

Sensors are used to trigger events likely to activate reconfiguration operations. We distinguish two kinds of Sensors:

1. *QoS and Resource Sensors*: They are generic components associated with QoS and Resource Probes. Their role is to inspect the evolution of observed parameters and to notify relevant changes. The behavior of such Sensors is quite simple: it consists in comparing the observed values with agreed-upon thresholds in order to detect changes in the observed entities. When a change occurs, the Sensor feeds back a corresponding event to the appropriate components.

2. *External-Event Sensors*: They are in charge of monitoring external events requiring a specific implementation. As an example, a Sensor may implement a conferencing manager listening for new connections and notifying the arrival of new participants. A second example would be a Sensor associated with the graphical user interface that sends relevant events.

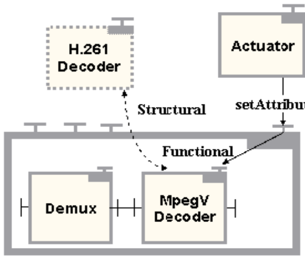


Fig. 4. Examples of reconfigurations

3.2.4 Actuators

Reconfiguration actuators are primitive components responsible for the execution of reconfiguration actions. Actuators react to events by performing required operations on the appropriate components. Each reconfiguration action on a component is performed through its attribute control interface (i.e. by modifying one of the component's attributes). This means that the Actuator has a generic behavior whatever the targeted components. It belongs to the component implementation to decide how to interpretate modifications of its attribute values. We distinguish three kinds of reconfiguration:

- *Functional Reconfigurations*: The most basic form of reconfiguration consists in changing the functional attributes of a primitive component belonging to the application. Reconfigurations may target key attributes in order to tune the produced media stream. As illustrated in Figure 4, the *Decoder* composite may delegate/forward modification of one of its attributes (e.g. 'quality') to its *MPEG-Video Decoder* sub-component, which performs the effective operation (and change the quality of the produced stream). Although such operations only affect media primitives, their impact on the media type agreed during initial bindings of this component defines two cases:
  1. In a first case, the targeted attribute does not affect the media type and therefore, this operation does not interrupt the execution of the application.
  2. In a second case, the targeted attribute affects the media type (e.g. a modification of the video resolution attributes or the representation format). This operation may require unbinding and re-binding the involved components according to rules explained in section 3.2.1.
- *Structural Reconfigurations*: This second form of reconfiguration concerns the modification of a composite's structure, being built-up of a set of sub-components. For instance, a modification of attribute 'encoding-format' of composite *Decoder*

leads to replace the *MPEG-Video* decoder with an appropriate one (*H.261 Decoder* in Figure 4). In general, this operation involves several steps illustrated in Figure 5:

- The *Activation* step includes the detection of changes (Probes/Sensors) and the decision of the execution of reconfiguration operations (Actuators).
- The *Pre-reconfiguration* step encompasses all tasks that can be performed without application interruption, among them: creating new components and setting their initial attributes. Stopping the application may be delayed in order to reduce the application black-out time.
- The *Blackout* step represents the period during which the application is stopped. This is necessary to unbind, remove and/or insert, and bind components.
- The *Post-reconfiguration* step encompasses all tasks that can be made after application restart, among them: deletion of old components and resetting of key attributes.

• *Policy Reconfigurations*: In the third form, reconfiguration actions target reconfiguration components themselves. Some of such reconfigurations are quite similar to functional reconfiguration and involve the modification of key properties of Probes, Sensors and Actuators. Examples consist in changing probing periods, tuning observation thresholds, modifying operations and operand values of reconfiguration actions, etc. Other reconfigurations may target the execution of these components by invalidating reconfiguration actions or activating/deactivating sensors.

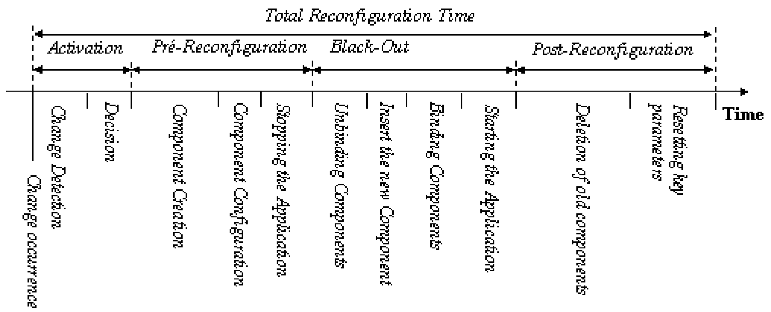


Fig. 5. Execution sequence of structural reconfiguration

4 Use Case: Mobile Multimedia Applications

PLASMA has been entirely implemented in C++ under Microsoft Windows 2000 using the Microsoft .Net 2003 development platform. Multimedia processing and networking functions are based on the Microsoft DirectShow toolbox. Several application scenarios have been successfully released, among them the SPIDER application, a media transcoding gateway for mobile devices.

4.1 SPIDER Architecture

The SPIDER architecture assumes several multimedia sources made available in the network, providing content in various data encoding formats and transmission protocols. Mobile users equipped with PDAs may access these sources; however they are limited to HTTP-based streaming and support exclusively MPEG-1 streams. The role of SPIDER gateways is to mediate between clients and servers by performing appropriate data conversions. A SPIDER node grants access to any multimedia source available in the network by means of data transcoding, transformation, protocol conversion, etc. Media streams are *receiver-driven*, i.e. client applications precise their media preferences (resolution, colors, bit-rate, etc.) as well as adaptation policies to adapt media streams when required.

A typical application scenario is as follows: the client application starts streaming video from a TV broadcast server (originally encoded in H.261 and transmitted using RTP). It requests a transcoding process which converts video content to MPEG with a resolution at 320x240. Knowing that potential increase of the transmission rate may overload its CPU and cause poor QoS (the gateway uses a Variable Bit-Rate encoding), it requests an adaptation policy which decreases the video resolution by 5 % whenever the transmission rate exceeds 512 Kbps. This adaptation algorithm is evaluated periodically (every 10 seconds in our experiment) in order to observe the behavior of the client application during different stages.

```
<TaskFlow id="Server" location="oxygene.inria.fr">
 <Task name="Input-Stream" id="C">
  <Attributes signature="InputAttributeController">
   <Attribute name="src" value="rtp://ozone.inria.fr:5000"/>
  </Attributes>
  <Binding id="b1" client="this" server="E" />
 </Task>
 ....
 <Task name="Video-Encoder" id="E">
  <Attributes signature="EncoderAttributeController">
   <Attribute name="format" value="32" />
   <Attribute name="resolution" value="320x240" unit="pixels" />
  </Attributes>
  <Binding id="b4" client="this" server="D"/>
 </Task>
     ...
 <Task name="Output-Stream" id="O">
     ...
 </Task>
 <Observation id="ob1" type="Resource" resource="id(O)@datarate">
  <event id="evt1" operator="exceeds" value="512" unit="kbps"/>
  <event id="evt2" .../>
 </Observation>
 <Action-set id="set1" condition="evt1">
  <Action operation="decrease" target="id(E)@resolution" operand="5"/>
 </Action-set>
               ….
</TaskFlow>
```

Fig. 6. A Dynamic ADL Description

Such information is conveyed from a client device to a SPIDER node as an ADL description sent using traditional session protocols. In our implementation, ADL descriptions are embedded within HTTP requests. Relying on PLASMA, each SPIDER node translates the ADL description into the suited multimedia adaptation service. Figure 6 shows the ADL description corresponding to the previous scenario.

The application architecture consists of a set of *Tasks* expressing high-level multimedia functions implemented by media composites. Each of them has a collection of attributes that precise its functional properties and its relationships with other Tasks (i.e. media bindings). It results in a task-graph representing the data processing sequence. In our scenario, the application is composed of four Tasks: an Input-Stream, a Video-Decoder a Video-Encoder and an Output-Stream. It receives an original RTP stream (see *src* attribute), decodes its content, encodes the result in MPEG-1, and transmits the result using TCP.

Reconfiguration policies are expressed in terms of *Observations* and *Action-sets*. Observations represent Probes and can be related to Task attributes or to resources (QoS or Resource Probes). Each observation defines one or more events reflecting violations of thresholds associated with observed parameters (i.e. Sensors). Here, a observation is associated with the data rate of the Output stream. Action-sets define one or more actions manipulating attribute values (*target* attribute).

4.2 Performance Evaluation

In our scenario, we used as SPIDER gateway a PC running Windows 2000, equipped with a 2 GHz Pentium 4 processor, 256 MB of Memory and 100 MB/s LAN. On the client side, we used a Compaq IPaq PDA equipped with a XScale Processor at 400 MHz, 64 MB of memory and 11 MB/s 802.11b WLAN. The performance evaluation concerns the cost of reconfiguration operations and their impact on performance of client device.

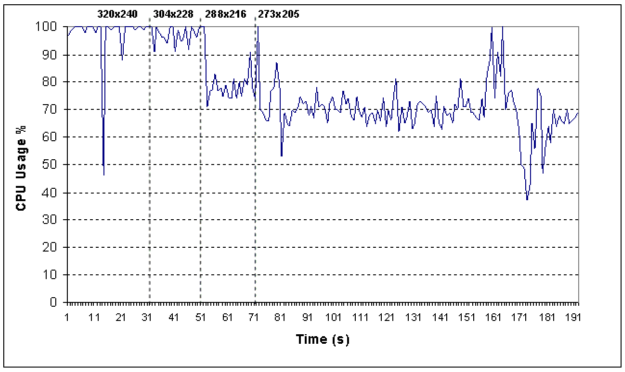
4.2.1 Performance of Structural Reconfiguration

The first reconfiguration occurrence requires a structural reconfiguration which adds a Video-Resizer component. The whole reconfiguration time is about 36 ms (milli-seconds). The major part is devoted to the pre-reconfiguration step with 24 ms, required for the instantiation of the new component. The blackout time is about 11 ms, spent in order to insert the Resizer component. Post-reconfiguration operations take about 200 s (micro-seconds). We deduce from these results that performing the pre-reconfiguration step before stopping the application significantly minimizes the cost of structural reconfiguration, in our case by 68 % assuming that a naive implementation would have required application be stopped during the whole time.

Subsequent reconfiguration operations on video resolution are only functional reconfigurations on the Resizer components. As these reconfigurations affect the output media type (the resolution), it requires re-binding (unbind, property modification and rebind) of components that are placed after the Resizer component in the processing graph. Setting component attributes takes 67 s. The blackout time is 331 s, where 244 s are necessary to re-bind components. Despite this short blackout, the whole reconfiguration time is low and does not introduce a significant overhead (i.e. about 398 s).

4.2.2 Impact on QoS and Resource Usage

In order to evaluate the impact of reconfiguration on the QoS and resource usage on the client device, we measured the rate of video display (in frames/seconds) observed by the application and the CPU. Figure 7 reports results. Vertical dotted lines mark the occurrences of reconfigurations, detected as a change in the video resolution. In the first time segment, video is displayed at less than 10 fps due to a CPU usage at 100 %. This is due to the fact that application receives a large amount of data and in addition, it must resize video frames in order to fit its display limitations. This operation requires additional processing causing the application to drop frames in response to CPU overload. The first reconfiguration reduces the video resolution to 304x228 and therefore, increases the frame rate to 17 fps. However, the CPU load is kept at 100 % as the transmission rate remains high. Finally, the resolution is reduced to 273x205, resulting in a frame rate at 25 fps and a CPU load at 70 %. These results show on one hand that reconfigurations don't have a negative impact since the reconfiguration blackout time is almost transparent to the client application. On the other hand, they significantly improve QoS and resource usage on the client.



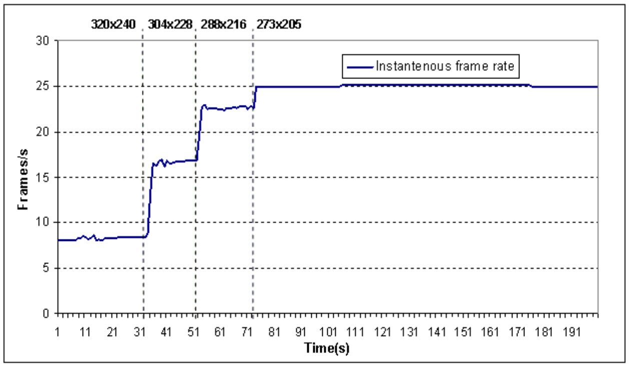


Fig. 7. Impact of reconfiguration on QoS and resource usage

5 Conclusion

This paper has presented PLASMA, a component-based framework for building self-adaptive multimedia applications. To reach this goal, PLASMA relies on an advanced component model whose main feature is the recursive composition of applications. Relying on this feature, a hierarchical reconfiguration management is introduced at different levels of hierarchy. This allows releasing a large scope of reconfiguration independently from the application structure and thus, it offers a flexible approach to adaptation. Our experimental study has shown that component-based adaptation as in PLASMA provides a good trade-off between flexibility and performance. Indeed, it does not introduce a high overhead and can significantly improve QoS and resource usage of multimedia applications.

References

1. G. Blair, L. Blair, V. Issarny, P. Tuma, A. Zarras. The Role of Software Architecture in Constraining Adaptation in Component-based Middleware Platforms. Middleware Conference, April 2000.
2. E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma and J-B. Stefani. An Open Component Model and its Support in Java. International Symposium on Component-based Software Engineering, May 2004.
3. A.P. Black, and al. Infopipes: an Abstraction for Multimedia Streaming. In Multimedia Systems. Special issue on Multimedia Middleware, 2002.
4. E. Cecchet, H. Elmeleegy, O. Layaïda and V. Quéma. Implementing Probes for J2EE Cluster Monitoring. In OOPSLA Workshop on Component and Middleware Proformance, Vancouver, October 2004.
5. L.S. Cline, J. Du, B. Keany, K. Lakshman, C.Maciocco, D.M. Putzolu. DirectShow RTP Support for Adaptivity in Networked Multimedia Applications. IEEE International Conference on Multimedia Computing and Systems, 1998.
6. Microsoft: DirectShow Architecture. http://msdn.microsoft.com/directx 2002.
7. D. Duke and I. Herman. A Standard for Mulimtedia Middleware. ACM International Conference on Multimedia. 1998.
8. T. Fitzpatrick and al. Design and Application of TOAST: An Adaptive Distributed Multimedia Middleware. International Workshop on Interactive Distributed Multimedia Systems, 2001.
9. X. Fu and al. CANS: Composable, adaptive network services infrastructure, USITS 2001.
10. H. Schulzrinne and al. RTP: A Transport Protocol for Real-Time Applications, 2003.
11. D. Sisalem and H. Schulzrinne. The loss-delay based adjustment algorithm: A TCP-friendly adaptation scheme. Proc of NOSSDAV '98, July 1998.
12. B. Stiller, C. Class, M. Waldvogel, G. Caronni, and D. Bauer, A Flexible Middleware for Multimedia Communication: Design, Implementation, and Experience," IEEE Journal on Selected Areas in Communications, September 1999.
13. Sun: Java Media Framework API Guide. http://java.sun.com/products/javamedia/jmf/ 2002.
14. L.A. Rowe, Streaming Media Middleware is more than Streaming Media. International Workshop on Multimedia Middleware, October 2001.

15. M. Lohse, M. Repplinger, P. Slusallek, An Open Middleware Architecture for Network-Integrated Multimedia, Joint IDMS/PROMS workhop 2002.
16. K. Nahrstedt, D. Wichadakul, and D. Xu. Distributed QoS Compilation and Runtime Instantiation. IEEE/IFIP International Workshop on QoS 2000.
17. S. McCanne and V. Jacobson. VIC: A flexible framework for packet video. ACM Multimedia Conference, 1995.
18. Z. Morley and al. Network Support for Mobile Multimedia using a Self-adaptive Distributed Proxy, NOSSDAV-2001.
19. D.G.Waddington and G.Coulson, A Distributed Multimedia Component Architecture, 1st International Workshop on Enterprise Distributed Object Computing, October 1997.
20. D. Wichadakul, X. Gu, K. Nahrstedt, A Programming Framework for Quality-Aware Ubiquitous Multimedia Applications, ACM Multimedia 2002.

Transformation Composition Modelling Framework

Jon Oldevik

SINTEF Information and Communication Technology,
Forskningsveien 1, 0373 OSLO, Norway
jon.oldevik at sintef.no
http://www.sintef.no

Abstract. When applying transformation technologies in an enterprise, there will be a need for supporting compositions of different kinds of transformations to support a development process. An example is a chain of transformations that supports a process of going from requirements to use cases, from use cases to a PIM architecture model, further to a platform specific model and finally implementation code. Some transformation steps may also involve human intervention, e.g. in a refinement of the PSM model, or a detailing of the use case model. This work in progress paper investigates how the atomic transformation viewpoint can be enhanced with support for transformation compositions, to support model driven enterprise process needs. This is done by introducing a modelling framework for composed transformations, based on a hierarchy of transformation types, some of which represent simple atomic transformations, others that represent complex transformations.

1 Introduction

Model transformation is an essential ingredient in model-driven development. In order to support the automation of system development, standards for transformation is emerging in the model-driven development community, driven by the Object Management Group (OMG). The forthcoming transformation standards such as the OMG MOF Query/View/Transformation (QVT)[1] is a good baseline for leveraging the model driven processes of enterprises. When applied in real use, needs will emerge to support the development scenarios in an enterprise, which is often complex and involves a set of integrated modelling, transformation, and validation tasks. An example development process, involving specification of requirements, use cases, architecture models, platform specific models and generation of code, will potentially involve a number of automated or manual transformation tasks, as well as analysis tasks such as model consistency checks.

This paper investigates how to support compositions of transformations that provide the needs that might occur in a model driven development scenario using UML 2 modelling techniques.

2 Transformation Composition Modelling Strategy

The transformation composition modelling strategy aims to support the construction of complex transformations that uses other atomic transformations. Assumedly, for a given enterprise, there will exist a number of defined transformations, which do simple or complicated transformation from a single domain to another.

These transformations might be combined to support parts of, or a complete, development process for the enterprise. The basis for the transformation modelling strategy is a number of transformation types which allows transformations to be combined. A transformation can involve for example a model to model transformation using QVT (or other transformation technology), a model refinement, which is done manually, and model to text transformations.

Doing a transformation in itself might not always be sufficient. A transformation imposes relationships that should be maintained and checked. Changes to models might require transformations to be reapplied. Resulting models might need to be check for consistency and validity. These kinds of analysis activities may also be part of a broader transformation framework.

2.1 Transformation Types

The heart of the transformation framework is the transformation types, which define transformation types with different natures. A *GenericTransformation* represents the common aspects of all types of transformation (Figure 1).

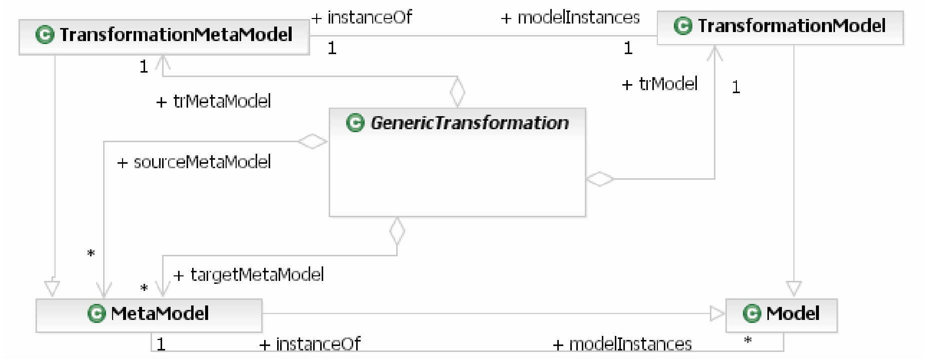


Fig. 1. Generic Transformation Structure

A GenericTransformation defines the overall structure and behaviour of all transformations. It is associated with a set of source and target metamodels. It has a transformation metamodel and a transformation model. During runtime, it consumes input (which may be a model) and produces output (which also may be a model).

The input and output provided by a *GenericTransformation* can be of different kind. They may be models, in case of a model to model transformation. Each of them may also be some kind of text (e.g. code), in case of a model to text or text to model transformation.

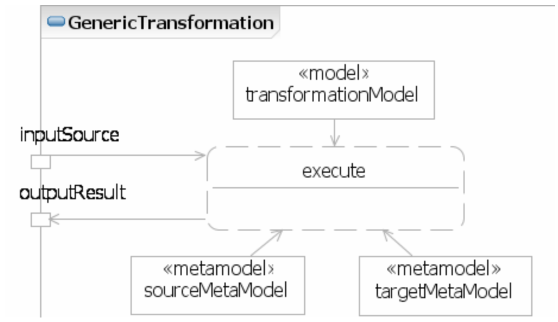


Fig. 2. Execution structure of a GenericTransformation

The *GenericTransformation* is specialised into different types of transformations, each of which has a special purpose. The framework is open for additional extensions that provide tailored transformation types (Figure 3).

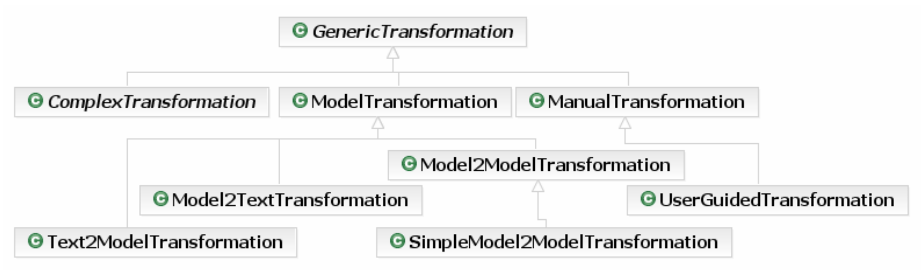


Fig. 3. Transformation Types Hierarchy

The transformation hierarchy defines the following specialisations:

- ModelTransformation, which represents automatic transformations involving models either as source or target or both.
- Model2ModelTransformation, which represents a transformation where involving models as both source and target of the transformation. It may include several source or target models.
- SimpleModel2ModelTransformation, which is a special case of Model2Model Transformation, represents model transformations with only one source and one target model.
- Model2TextTransformation, which represents a transformation from a set of source models to text output.
- Text2ModelTransformation, which represents a transformation from a text to a set of models.

- ManualTransformation, which represents transformations that involves human intervention (i.e. not supported by transformation tools), e.g. manual refinement of a model.
- UserGuidedTransformation, a special kind of manual transformation which is guided by user advice (input), but is executed by a tool.
- ComplexTransformation, which represents more complex transformations (Figure 4)

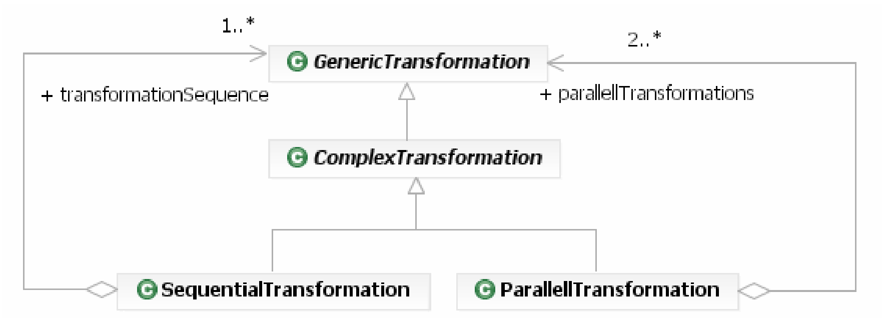


Fig. 4. Complex Transformation Types

The *ComplexTransformation* represents transformations that involve several simpler (atomic) transformation tasks. Two types of complex transformations are identified:

- ParallellTransformation, which represents transformations where the referenced transformations can/must be executed in parallel. A parallel transformation references two or more transformations, which are part of the parallel semantics.
- SequentialTransformation, which represents transformations where the referenced transformations must be executed in sequence. A sequential transformation references one or more transformation, which are part of the sequential semantics.

The goal of the *ComplexTransformation* types is to be able to construct useful composite transformations, which basically invokes already existing, well-defined transformations.

2.2 Building Composite Transformations

The composition of transformations requires that there already exists some reusable transformations to compose from. An assumption is taken that a library of existing transformation is readily available. These are described in terms of simple UML 2 activities, which consume input and produces output. The transformations are represented as activity nodes, with input objects (models and metamodels) and output models. Object Flows are used to model the flow of models and metamodels used by the transformation. Activity Parameters denote the consumed and produced input by a transformation.

Figure 5 depicts two transformations; a model to model transformation, going from a Use Case model to a Platform Independent Model (PIM); a manual transformation, refining a PIM model. The transformation themselves are not detailed. They are references, and can be e.g. a QVT specification.

Figure 6 depicts an example of a composition. It is a *SequentialTransformation*, which contains references to the transformations defined in Figure 5, plus an additional PIM2PSM transformation, and a parallel transformation containing two text transformations. Each of the referenced transformations may themselves be more composites, thus allowing for arbitrary complexity of transformations.

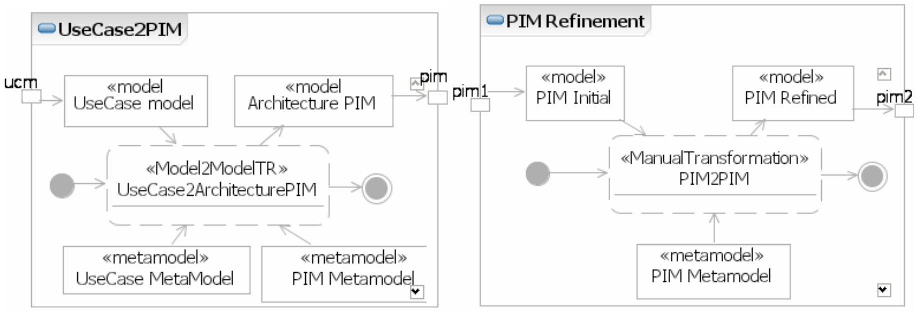


Fig. 5. Use Cases to Platform Independent Architecture Model (PIM)

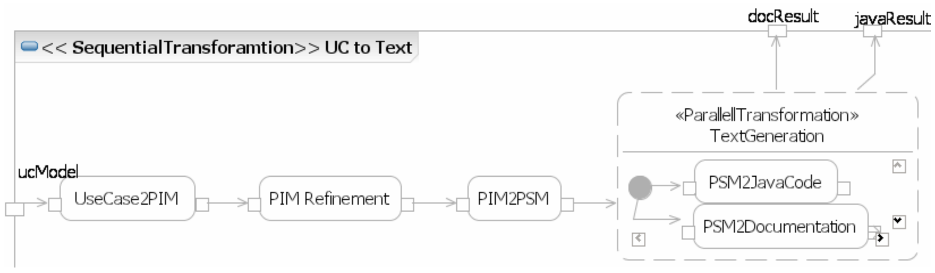


Fig. 6. Composite Transformation – Use Case through Text

The process of executing a complex transformation is basically an orchestration task, i.e. making sure that each transformation is executed in its rightful order. In addition, transformation processes may require different kinds of analysis to be done as part of the process. Analysis tasks are also a necessary part of model-driven development chains. Thus, specific tasks handling various pre-defined or tailored analysis of models or text, should have its place in a modelling and transformation framework. This includes aspects such as model conformance, model completeness, traceability, validation, and other analysis tasks relevant for the models or text artefacts consumed or produced. This could be incorporated at a modelling level to support a more holistic model-driven approach.

2.3 Relationship with QVT

The forthcoming QVT language [1] can be used to specify complex transformations in its lexical notation. It also supports composition of transformations through various reuse mechanisms, such as extension or black-box reuse of transformation libraries.

Given a set of basic transformations that stands on their own, it is possible to create a composite transformation, e.g. through the *access* mechanism of QVT.

```
transformation UC2PSM (in ucm : UseCaseMM, out psm: PSM):
access transformation UC2PIM (in ucm : UceCaseMM, out pim : PIM);
access transformation PIM2PSM (in pim : PIM, out psm: PSM);
main () {
   var pimResult := UC2PIM(ucm).transform();
   psm := PIM2PSm(pimResult).transform();
}
```

Combined with control constructs (like conditional branches), structured transformation compositions can be created. The graphical QVT syntax might be used for similar constructions, although the current graphical notation is tuned towards specifying relations between the concepts of a single transformation. A UML2-oriented approach with composite activities (or even composite structures) can leverage specification of higher-order transformations.

3 Related Work

In [2], a framework for composition of transformation is proposed, which defines a pattern for composite transformation and a lexical language for defining composites, which handles configuration (execution ordering) of transformation components. This framework does not propose any UML 2 coupling or relate to QVT. In [3], the side transformation pattern is introduced to provide a means of describing combinations of reusable transformations with particular focus of coping with application-specific metamodel incompatibilities. The approach uses the graphical UMLX notation, resembling the graphical QVT notation. Within web service orchestration, choreography and composition [4], a lot relevant of work and technology exist that is pertinent for describing reusable compositions of transformations, such as the Business Process Modelling Notation (BPMN), the Business Process Execution Language (BPEL).

4 Summary and Conclusion

This work in progress paper has described ongoing work in the ModelWare project (IST Project 511731)[*], concerning how to handle complex structures of transformations, using a model-based approach for composition of atomic transformations. The continuation of the work will focus on elaborating the modelling framework and support it with tools which are integrated with standards like QVT.

[*] MODELWARE is a project co-funded by the European Commission under the "Information Society Technologies" Sixth Framework Programme (2002-2006). Information included in this document reflects only the author's views. The European Community is not liable for any use that may be made of the information contained herein.

References

1. QVT-Merge Group, Revised submission for MOF 2.0 Query/Views/Transformations RFP version 2.0, OMG document id ad/2005-03-02, http://www.omg.org/cgi-bin/apps/doc?ad/05-03-02.pdf
2. Raphaël Marvie, *A Transformation Composition Framework for Model Driven Engineering,* LIFL 2004n10, November 2004
3. Willink, Harris, 2004, The Side Transformation Pattern, paper at the Software Evolution through Transformations (SETra) 2004
4. Peltz C., *Web Service Orchestration and Choreography*, Web Service Journal Feature, July 2003

Proximity-Based Service Discovery in Mobile Ad Hoc Networks

René Meier, Vinny Cahill, Andronikos Nedos, and Siobhán Clarke

Distributed Systems Group, Department of Computer Science,
Trinity College Dublin, Ireland
{rene.meier, vinny.cahill, andronikos.nedos,
siobhan.clarke}@cs.tcd.ie

Abstract. Existing approaches to service discovery have been developed primarily for environments with a fixed network backbone and typically rely on centralized components being accessible to potential service clients at any given time. The characteristic lack of a designated service infrastructure in combination with the highly dynamic nature of the underlying network topology renders such discovery mechanisms unsuitable for mobile ad hoc environments. This paper presents an approach to the discovery of ad hoc services that exploits the fact that the relevance of such services is often limited to specific geographical scopes. Service providers define the areas (proximities) in which their services are available. Clients register interest in specific services and are subsequently informed whenever they enter a proximity within which these services are available. Since ad hoc services can be stationary or may be moving with the location of their mobile providers our approach supports discovery of services with fixed locations as well as of those that migrate with their providers. Our approach has been implemented as a push-based proximity discovery service and its evaluation demonstrates that it is well suited for highly dynamic networks as it maintains neither routes nor overlay network topologies.

1 Introduction

Many mobile applications can be characterized as collaborative in the sense that mobile nodes use the wireless network to interact with other mobile nodes that have come together at some common location. Collaborative nodes typically come together in some area, establish associations with other collaborative nodes dynamically, and make use of common services already available in that locality or provided by members of the group. The members of such a group may migrate together, like a fleet of vehicles or the participants in a guided tour. Although these applications may use infrastructure networks, they will often use ad hoc networks since they are immediately deployable in arbitrary environments and support communication without the need for a separate infrastructure. This collaborative style of application may be useful in the ubiquitous [1] and sentient computing [2] domain allowing loosely-coupled, highly-mobile components to communicate and collaborate in a spontaneous manner.

Entities that comprise this style of application are inherently mobile and may move together and apart over time. Service providers and service clients characteristically come together at a certain location, use services, and later come together at a different location to use other services. Hence, providers require a means to advertise their services and clients need be able to locate services of interest depending on their current location.

Existing approaches to service discovery [3-5] have been developed primarily for environments with a fixed network backbone and typically rely on centralized components being accessible to potential service users at any given time. The characteristic lack of a designated service infrastructure in combination with the highly dynamic nature of the underlying network topology renders such discovery mechanisms unsuitable for ad hoc environments. This paper presents a decentralized mechanism for the discovery of services in ad hoc environments that exploits the fact that *the relevance of such services is typically limited to specific geographical scopes*. Example services from the traffic management domain might include an accident warning service provided by a crashed car to approaching vehicles and a warning service provided by an ambulance rushing to an accident site to inform nearby vehicles to yield the right of way. Providers define the geographical scopes, called proximities, in which their services are available. Our Proximity Discovery Service (PDS) allows such providers to advertise their services in proximities, thereby bounding advertisements to the areas in which the services are relevant. Interested clients are informed whenever they enter a proximity where these services are available. The PDS therefore enables clients to dynamically discover available services (and their proximities). Clients subsequently use the discovered information to establish logical connections to the associated providers. The connections between the entities residing in a particular proximity are then used by providers to deliver their services hence, *allowing clients to use them at the specific location where they are valid*. This concept of proximity-based service discovery enables collaborating entities to come together at a certain location and to spontaneously discover the services available at that location.

Clients must register interest in specific services with the PDS in order to be informed whenever they enter a proximity where these services are available. Such clients may move from one proximity to another without re-registering their interest. Thus, registrations are persistent and will be applied transparently whenever a client enters a new proximity. This implies that a registration to a specific service type applies to all proximities in which these services are provided even though the client may only use a subset of these services at any time. A single registration may result in a service by different providers in multiple proximities being discovered. For example, a client representing a vehicle that registers interest in an accident warning service will be informed whenever it enters the proximity of such a service.

Services can be stationary or may be moving with the location of their mobile providers. A stationary service is attached to a fixed point in space whereas a mobile service is mapped to a moving position represented by the location of its mobile provider. Hence, a mobile service moves with the location of the provider to which it has been attached. An example of such a mobile service might include an information service provided by an entity representing an ambulance on a call. This implies that proximities may be moving with their service. The PDS therefore supports *discovery* *of services (and proximities) with a fixed location as well as of those that migrate with their providers*.

The PDS has been implemented as part of an event-based middleware for wireless ad hoc environments [6] where it is used for the discovery of relevant event producers. However, the discovery paradigm of the PDS can be used as a means for locating arbitrary services. The evaluation of the PDS demonstrates that using proximity to bound multicast-based service discovery is well suited for highly dynamic networks as it limits the complexity of the underlying dynamic network topology without introducing overhead for maintaining discovery routes and overlay network structures.

The reminder of this paper is structured as follows: Section 2 surveys related work. Section 3 presents our architecture for discovering services in mobile environments and describes how this architecture maps to the underlying ad hoc network. Section 4 outlines our evaluation of the PDS. Finally, section 5 concludes this paper by summarizing our work.

2 Related Work

The concept of service discovery is well established in distributed systems [7], since networked entities need to locate available remote resources in order to use them. Work on service discovery in mobile ad hoc networks focuses on using decentralized architectures to overcome the limitations of traditional discovery mechanisms, such as SLP [3], Jini [4], and UPnP [5], that rely on fixed infrastructure. Research in service discovery architectures for such mobile environments can be classified into lower-layer service discovery protocols [8-11] and higher-layer service platforms [12-14].

Service discovery protocols place emphasis on efficiency and distributed infrastructure while service platforms focus on providing a middleware layer that enables applications to use a service oriented programming model. The PDS bridges this gap by providing a programming model for service discovery based on location while exploiting a proximity-based multicast protocol that limits the complexity of the underlying ad hoc network topology without introducing overhead for maintaining routes and overlay network structures.

Research in the area of discovery protocols has produced encouraging results with some protocols exploiting location to achieve distributed discovery, while others rely on the formation of a virtual backbone of nodes which can subsequently act as service brokers. Protocols that use location for service discovery include the Grid Location Service [10] and the Distributed Location Management [9] protocol. Despite primarily being protocols that enable the distributed lookup of any node's location, in the context of service discovery, location can be considered as just another service. The use of location in such protocols offers nodes a kind of "topological awareness" and aids the distributed execution of their respective algorithms. The PDS differs in its use of location, as the above protocols require stricter assumptions such as knowledge of a geographic grid that must be shared by all operating nodes. The PDS has no such requirement and uses location as a constrain to enhance scalability and relevancy.

Protocols like [11] and [8] provide a fully distributed service discovery mechanism based on using selected mobile nodes that form a virtual backbone. The nodes in the backbone subsequently act as service brokers, permitting service registration and thus, facilitate service discovery. Such protocols have proved to perform well under simulation compared to broker-less service discovery mechanisms that use multicast or anycast. However, they require specialized backbone formation and communication protocols which, coupled with the cost incurred by maintaining the virtual backbone, can result in reduced performance under different mobility scenarios. In contrast, the PDS requires no maintenance phase other than maintaining the multicast tree and increases efficiency by limiting packet forwarding to the proximity area.

Konark [14], GSD [13], and DEAPspace [12] provide service discovery platforms. Konark is a service discovery mechanism that is based on a hierarchical classification of services. Its architecture supports both push and pull style discovery, but is limited to locally scoped multicast. In comparison, the PDS hides the complexity of the underlying communication system by providing an asynchronous advertise/discover programming model to the application layer that is also independent of the structure of service representation. This has the extra benefit of allowing the PDS to work with multiple service representations. GSD uses caching and semantic matching of services to efficiently discover services in a distributed manner. GSD uses a predefined ontology to group similar services and forwards service requests efficiently by exploiting this grouping. The PDS uses a more sophisticated forwarding mechanism that relies on the notion of proximity rather than hop count. DEAPspace has been designed for pervasive environments and shares the same discovery paradigm as the PDS, namely push based discovery, but has a more narrower scope than the PDS as it is directed towards personal area networks. It was designed for small devices and provides energy efficient mechanisms for message dissemination. However, there are no clear guidelines as to how applications interact with DEAPspace. The PDS on the other hand, supports a concise application programming interface and its use of proximity implies support for single-hop and multi-hop ad hoc networks.

3 The Architecture for Service Discovery

The design of the architecture for service discovery is motivated by the hypothesis that there are services that are more likely to be used by clients once they are in close proximity. This means that the closer clients are located to a provider the more likely they are to be interested in the services that it offers. Significantly, this implies that services are relevant within a certain geographical area surrounding a provider.

The architecture is inherently distributed in order to support ad hoc environments. As shown in Fig. 1, the middleware that includes the PDS is exclusively collocated with application components and depends neither on centralized or intermediate components nor on the presence of a designated infrastructure. Every mobile device hosting the middleware therefore offers identical capabilities to its application without accessing remote components. Applications may connect a variable number of providers and clients to the middleware on each mobile device, thereby allowing individual devices to offer and to use one or more services. Each of these entities regulates its use of the PDS by exploiting the concepts of advertisement and registration. Providers advertise the proximities in which their services are available.

Clients register interest in specific services in order to be informed whenever they enter a proximity where these services are available until they un-register. The PDS augments these well-known and widely-used concepts in order to support scoped service discovery and ultimately mobility. The proximity information announced by providers is exploited primarily to establish communication relationships between mobile entities while registrations are used locally to map client interests to the currently available services.

The PDS has been designed to discover services for entities that can be either stationary or mobile. This implies that the PDS as well as the entities hosted by a particular mobile device are aware of their geographical location at any given time. The middleware therefore includes a Location Service (LS) that uses sensor data to compute the current geographical location of its host device and entities. To suit outdoor applications, for example in the traffic management domain, the middleware exploits a version of the LS that uses a GPS satellite receiver to provide latitude and longitude coordinates.

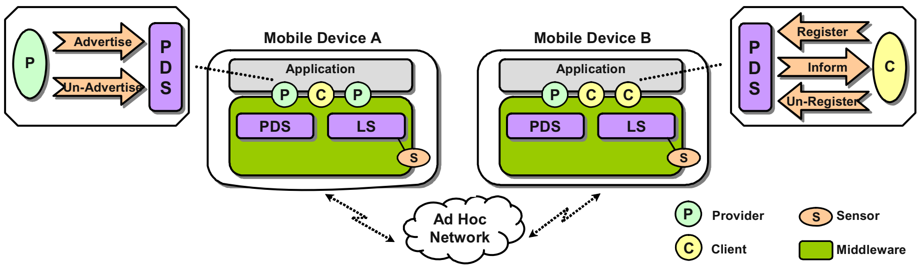


Fig. 1. Discovery service architecture

3.1 Stationary and Mobile Services

The PDS supports both stationary and mobile entities and consequently must be able to handle stationary and mobile services and their proximities. The proximity definition below shows that this notion of proximity is defined firstly by the covered area, which is described as a geometric shape with associated dimensions and a reference point that is relative to this shape, and secondly by a naval location. The reference point of a stationary proximity is attached to a naval represented by a fixed point in space whereas the reference point of a mobile proximity is mapped to a moving naval, which is characteristically represented by the location of a specific mobile provider. Hence, a mobile proximity moves with the location of the provider to which it has been attached. For example, a group of vehicles heading in the same direction may cooperate to form a platoon in order to reduce their consumption of fuel. These vehicles might use a mobile service provided by the leading vehicle. The proximity of such a service might be attached to a naval represented by the position of the leader thereby moving with its location. Proximities may be of arbitrary shape and may be defined as nested and overlapping areas. Nesting allows a large proximity to contain a smaller proximity subdividing the large area.

Proximity = [Area (Shape, Dimensions, Reference Point), Naval]

3.2 Identifying Services

There are two essential issues that need to be addressed when discovering services. Firstly, an addressing scheme for uniquely identifying services is required and secondly, a means for clients to obtain the correct identifiers needs to be provided. An approach to addressing these issues, based on statically generating a fixed number of unique and well-known identifiers, has been described by Orvalho et al. [15]. Another approach might involve using a centralized lookup service for generating and retrieving identifiers. However, neither of these approaches suffices for applications that accommodate a dynamically changing number of services and depend on an inherently distributed architecture.

The PDS exploits a decentralized addressing scheme based on a Distributed Hash Table (DHT) in which identifiers representing services can be computed from discovered service descriptions. Each combination of service name and proximity (shape dimensions, reference point, and naval location) is considered to be unique throughout a system assuming that there is no justification for providers to define multiple identical service name and proximity pairs for different services. A textual description of such a pair is used as stimulus for a hashing algorithm to dynamically generate hash keys that represent identifiers using node local rather than global knowledge. Clients compute the corresponding identifier upon discovery of a service and subsequently use these identifiers to connect to services.

This distributed addressing scheme replaces the kind of centralized approach traditionally used for identifying services of interest and therefore represents a key enabling mechanism for the inherently distributed architecture of the PDS. It enables mobile devices to recognize services of interest and to *locally* compute identifiers from serialized service descriptions.

Computing Service Identifiers. The PDS uses a hashing algorithm that generates 24 bit service identifiers from variable length character strings. The implemented algorithm is based on a combination of a hash function for such stimuli proposed by [16] with the use of a hash function multiplier [17] in order to reduce the chance of collisions. Fig. 2 depicts the structure of the character strings that describe services with rectangular and circular proximities. Such strings comprise the service name and the particulars of the associated proximity. Proximities are described by the dimensions and reference points of their shapes and by the coordinates that specify the location of their navals. Note that the proximity of a mobile service always describes its initial naval location since its actual naval location might change over time therefore enabling clients to generate consistent identifiers.

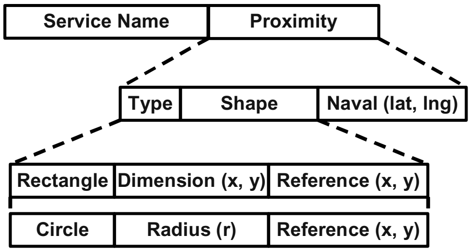


Fig. 2. Computing service identifiers

Collisions. Depending on a number of factors, including the quality of the hash function and the ratio of stimuli to potential identifiers, this approach might lead to colliding identifiers. Such collisions occur when two distinct stimuli x and y generate the same identifier. In other words, there exist stimulus pairs, such that x ! y, for which h(x) = h(y). Collisions may result in different services using the same identifiers. This does not affect a system provided that such services are used in different geographical scopes, i.e., that their proximities do not overlap. Overlapping proximities with colliding identifiers can lead to unwanted service messages being received by certain mobile devices. Such devices can be prevented from delivering unwanted service messages to their clients by a run-time type checking mechanism. Such a mechanism can detect and eventually discard these service messages. Hence, colliding identifiers may lead to additional use of communication and computational resources, but will not cause delivery of unwanted service messages and are in any case unlikely due to geographical separation.

3.3 Discovering Services

The PDS runs on every physical machine that hosts service users regardless of whether these local entities act either as providers or as clients, or indeed as both. The PDS uses beacons to periodically advertise relevant services (and the associated proximities) on behalf of providers. The discovery service advertises the service name and proximity pairs that have been defined by providers within the scope of a proximity. This implies that the location at which these advertisements are disseminated can change when the device hosting a PDS migrates and that the set of adjacent devices is likely to change as well. Moreover, this also implies that other stationary or mobile devices may need to forward advertisements for them to reach the boundaries of the proximity.

```
advertise(serviceName sn, proximity p)
un-advertise(serviceName sn, proximity p)
register(serviceName sn, discoveryHandler dh)
un-register(serviceName sn, discoveryHandler dh)

discoveryHandler {
  inform(serviceName sn, proximity p, status s) {
  serviceIdentifier = h(sn + p.toString())
  //use discovered service
  ...
  }
}
```

Fig. 3. The application programming interface of the PDS

Using the Discovery Services. Fig. 3 outlines the application programming interface supported by the PDS. These interface functions reflect the fact that the PDS is based on an implicit discovery model as they refer neither to explicit entities nor to designated components of any kind. Instead, the functions for advertising (un-advertising) and registering (un-registering) interest refer to a service name.

Providers and clients must use a common vocabulary defined by the application to agree on the name of a service. Such a `serviceName` typically describes a service type rather than referring to a service provided by a particular provider. This allows clients to discover services based on their availability (at a location) regardless of the specifics of their providers.

The PDS application programming interface also illustrates how providers advertise service proximities and how clients specify discovery handlers. Clients implement discovery handlers and subsequently register them. The PDS uses these handlers to inform clients of services that either have become available or have expired. Clients use `serviceName` and `proximity` pairs to generate service identifiers and, depending on the service `status`, commence or discontinue using the service. This approach enables clients to potentially register a specific discovery handler whenever they express their interest. Depending on their requirements, clients may therefore provide either a discovery handler for an individual service or for a set of services.

The Discovery Mechanism. The PDS allows its providers to advertise and its clients to discover relevant services according to the following discovery mechanism:

1. Initially, a proximity discovery service recognizes the services advertised by its local providers.
2. A proximity discovery service maintains a list describing all services that are relevant at its current location. A specific service description is discarded when the hosting mobile device leaves the proximity associated with this service description.
3. A proximity discovery service periodically broadcasts messages describing the services advertised by its local providers within the proximity of the advertised service relative to its current location. The broadcast period is application specific and hence, can be configured for each individual proximity discovery service.
4. A proximity discovery service receiving an advertisement adds a service description to its own list if it is currently located inside the associated proximity.
5. A proximity discovery service maintains a list describing the interests registered by its local clients.
6. Clients are informed whenever a service of interest is added to or discarded from the list of relevant services.

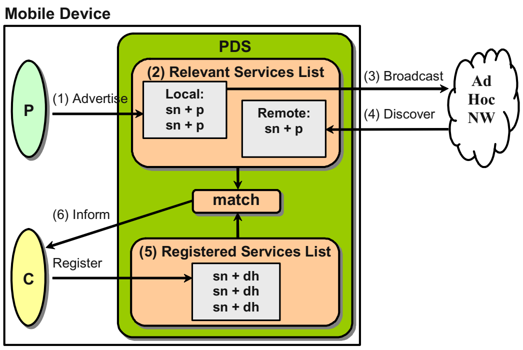


Fig. 4. The discovery mechanism

Fig. 4 illustrates the components of our approach to maintaining lists representing relevant services and services of interest respectively. The proximity discovery mechanism essentially comprises two algorithms, one for disseminating (locally-defined) service advertisements and another for handling (remotely-defined) service advertisements upon receiving them. Both algorithms operate on the relevant services list of a particular mobile device and collaborate in order to maintain the relevant locally and remotely specified services. Hence, relevant services lists are maintained according to geographical relevance and are independent of the set of issued registrations.

Advertising Services. The algorithm for broadcasting and to a certain extent maintaining service descriptions on a particular mobile device periodically traces through all service descriptions stored in a relevant services list in order to advertise the services that have been defined by local providers and to verify the geographical validity of service proximities. Hence, the algorithm essentially identifies stationary and mobile services with proximities that are relevant at the actual location of the mobile device and periodically advertises those locally-specified services.

Stationary services remain in a list as long as their scope includes the current location regardless of whether they have been specified locally or remotely. A service is removed from the list once the device has left the associated proximity. Removing a relevant service may lead to a change to the set of services available to local clients. Clients that have registered with such a service are informed of its cancellation. However, their interest, expressed by a related registration, remains in the registered services list.

The dynamic aspect of mobile services causes some variation in the means by which they are maintained and advertised. Mobile services specified by local providers always (by definition) include the actual location of the mobile device and migrate together with the device. Consequently, the migration of a mobile device does not cause mobile services to expire. These services can therefore be advertised without verifying their validity. In addition to enabling remote devices to discover mobile services, these advertisements provide a means for disseminating location updates describing the migration of mobile services. The naval of the proximity of every mobile service is therefore updated with the latest device location prior to being advertised. The validity of remotely-specified mobile services is verified when updates on their latest locations are received. This prevents validity checks using cached and therefore potentially obsolete (due to migration) location information.

Locating Services. The algorithm for handling received service descriptions adds newly discovered services to the relevant services list of any mobile device residing inside the proximity while known services, whose proximities no longer include the device's location, are removed from the list. Clients with a corresponding registration are informed of newly discovered services as well as of cancelled services. Other service descriptions comprise information that is either irrelevant at the current location or already known and are therefore discarded.

Remotely-specified services are handled similarly regardless of whether they describe stationary or mobile services. Both stationary and mobile services that are no longer relevant at the current location are removed from the repository. Mobile services are removed based on their latest naval location while being identified according to their initial naval location. This enables the PDS to identify mobile services even though their actual naval location changes.

3.4 Supporting Ad Hoc Networks

The PDS implements a location-based routing protocol that exploits proximity to control multicast-based flooding of the underlying ad hoc network [18]. This approach uses a well-known multicast group to provide a means for disseminating service descriptions in a one to many manner within the boundaries defined by a proximity without introducing extra overhead for maintaining routing information. Such routing information characteristically needs to be updated more frequently with increasing speed of mobile devices and thus, approaches that attempt to maintain routing information are less suited for applications comprising highly-mobile entities [19].

Providers may define proximities for their services independently of the physical transmission range of their wireless transmitters with the underlying communication system routing messages from provider to client using a multi-hop protocol. Nodes residing within the boundaries of a proximity forward and deliver these messages while nodes located outside a proximity discard them without forwarding.

The concept of proximity provides an application level abstraction for bounding the message dissemination scope. However, proximities can also be exploited in order to optimize the delivery of specific messages. The PDS uses a provider's geographical location and radio transmission range in conjunction with the proximity of its services to determine whether to use single-hop or multi-hop messages. The PDS therefore supports both single-hop and multi-hop dissemination and typically uses cost efficient single-hop messages for advertising services in proximities that are likely to be covered by a provider's wireless transmission range and employs multi-hop messages only when transmitting advertisements beyond this range.

4 Experimental Results

This section evaluates the proximity-based approach to the discovery of ad hoc services that is proposed in this paper. The main objective of the experiments is to assess the cost of service discovery and more specifically, to assess how exploiting proximity limits advertisement forwarding without the need for control messages. The experimental results demonstrate that discovery cost for stationary services are comparable to those for mobile services and that cost neither depends on the speed of clients nor on the speed of the service provider. This evaluation therefore demonstrates that using proximity to bound multicast-based service discovery is well suited for highly dynamic networks as it limits the complexity of the underlying dynamic network topology without introducing overhead for maintaining discovery routes and overlay network structures.

The primary measurement of interest is an abstract quantity we refer to as *cost*. We assign a relative cost to the dissemination of a single advertisement. Cost describes the number of messages required when propagating an advertisement from a provider to the clients residing within its radio transmission reach and the forwarding of this message to clients beyond this range. Hence, cost depends on the number of clients residing within a particular proximity and provides a qualitative indication of the bandwidth required for discovering a service. For example, the cost of a provider disseminating an advertisement to three clients, each of which forwards the message that describes the service once, is described as 4; one message sent by the provider and 3 messages forwarded by the clients. This allows us to measure cost independently of the actual frequency of an advertisement while varying a number of application parameters. These parameters include the migration speed of the entities, the range of the proximity within which services are advertised, and the number of clients.

All evaluation experiments have been conducted by implementing the selected scenarios as prototypical service applications. Each of the providers and clients that comprise these applications is represented by an independent PDS instance and interacts with other entities using a real ad hoc network. The entities and their PDS instances are hosted by a number of notebook computers running the Microsoft Windows XP operating system on a 1GHz Intel Pentium III processor. Each machine is equipped with a Lucent Orinoco Gold WiFi PCMCIA card with a channel capacity of 11 Mbit/s providing the ad hoc network connection for inter entity communication. One or more entities may reside on a notebook computer. Entity locations and mobility are simulated throughout these experiments using the location service of their respective PDS instances while the hosting notebook computers were placed within ad hoc communication reach of each other. The radio transmission range T_R for each machine was simulated to be T_R = 200 meters. This ensures that an entity discards all communication messages received from entities located beyond T_R, even though the distance between the physical locations of their host machines is less than T_R. Multiple runs were conducted for each experiment and the data collected was averaged over those runs.

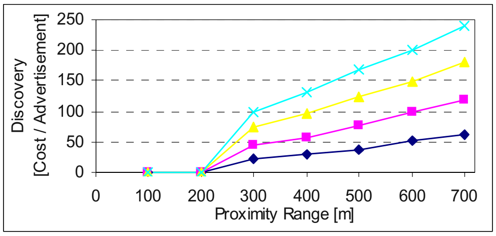
4.1 The Application Scenarios

We have selected two application scenarios from the traffic management domain for this evaluation. $Scenario_A$ models a broken down car providing a stationary accident warning service to vehicles within its vicinity. This scenario is set on a two way road with a provider acting as the broken down car advertising its service to clients representing the vehicles. These vehicles are randomly distributed on the lanes of the road. $Scenario_B$ models an ambulance providing a mobile warning service to nearby vehicles for them to yield the right of way. This scenario is set similarly to the previous scenario on a two way road and comprises an ambulance advertising its service to a number of randomly distributed clients representing vehicles. Both scenarios include a circular shaped proximity with a central reference point that is mapped to the location of the respective provider and involve a road section of 1400 meters with the provider being located at the center of the section. All clients move on this road section and their number defines the saturation of the area.

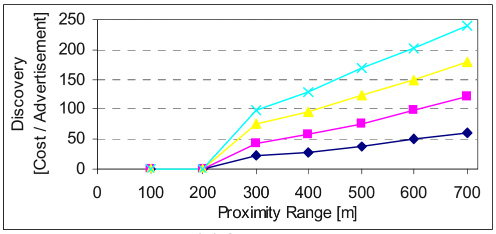
4.2 Results and Analysis

Fig. 5 (A) shows the discovery cost as a function of proximity range P_R. The respective proximities define the set of clients residing inside the area of interest that discover the service. These ranges have been selected to include services with proximities in which all clients can be reached using a single-hop radio transmission P_R ! T_R) as well as those that require multi-hop routing ($P_R > T_R$). The largest proximity covers the whole scenario area enabling all clients on the road section to discover the service. The results essentially demonstrate how proximities bound service discovery cost by bounding the number of clients that forward a certain advertisement. They show a significant difference when comparing discovery cost within the single-hop reach to the cost beyond this range.

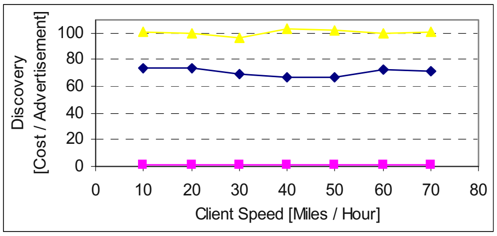
Beyond the single-hop range, all results show similar tendencies of increasing cost with expanding proximities and rising saturations as every client residing inside a certain proximity forwards advertisements. This illustrates that exploiting proximity enables the PDS to transparently select the appropriate protocol when disseminating advertisements. The PDS uses its cost efficient single-hop protocol for advertising services with proximities that are covered by the provider's T_R and employs the multi-hop version only when transmitting advertisements beyond T_R. Other discovery service platforms typically use either a single-hop protocol with propagation range limitations or a more expensive multi-hop protocol for both short and long range discovery. However, Fig. 5 (A) most significantly illustrates that the results recorded for $\mathtt{Scenario}_A$ and $\mathtt{Scenario}_B$ are virtually identical demonstrating that, given a similar configuration, the discovery cost for stationary services are comparable to those of mobile services.



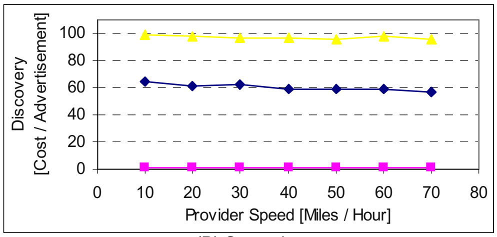
(A) Scenario_A



(A) Scenario_B



(B) Scenario_A



(B) Scenario_B

Fig. 5. (A) Discovery cost as a function of proximity range for saturations of 60, 120, 180, and 240 from bottom to top. (B) Discovery cost as a function of migration speed for proximities with T_R = 200, 400, and 600 meters from bottom to top and for a saturation of 120

Fig. 5 (B) depicts the service discovery costs recorded for $\mathtt{Scenario}_A$ and $\mathtt{Scenario}_B$ respectively as a function of migration speed. These results show the effect of migration speed and thus, of a dynamically changing network topology that reflects entity movements, on discovery cost. Similar results were recorded for $\mathtt{Scenario}_A$ where mobile clients discover a stationary service and for $\mathtt{Scenario}_B$ in which a mobile service provided by an ambulance moving along the road is being discovered by stationary clients representing stopped vehicles. These results essentially show that the cost is low within single-hop reach and increases with service proximities expanding beyond single-hop range. Significantly, the results recorded for $\mathtt{Scenario}_A$ and $\mathtt{Scenario}_B$ demonstrate that service discovery cost does neither depend on the speed of clients nor on the speed of the provider. This is due to the fact that the PDS exploits proximities to control multicast-based flooding and consequently does not introduce extra overhead for maintaining routing information that needs to be updated more frequently with increasing relative migration speed. The study of flooding-based multicast protocols for ad hoc networks presented by Lee et al. [19] presents similar conclusions and hence, argues that neither the number of transmitted messages nor the associated delivery ratio is a function of the relative speed of the interacting entities.

5 Summary and Conclusions

This paper presented a decentralized approach to discovery of services in ad hoc environments. The approach supports discovery of services that are stationary as well as those that move with the location of their mobile providers and exploits the fact that the relevance of such services is often limited to specific geographical scopes. Providers and clients of this style of service characteristically come together at a certain location, use a service, and later come together at a different location to use another service. Providers define the geographical scopes, called proximities, in which their services are available. Our Proximity Discovery Service allows such providers to advertise their services in proximities, thereby bounding advertisements to the areas in which the services are relevant. Clients are informed whenever they enter a proximity where these services are available. The Proximity Discovery Service therefore enables clients to dynamically discover available services. Clients subsequently use the discovered information to establish logical connections to the associated providers. The connections between the entities residing in a particular proximity are then used by providers to deliver their services thereby allowing clients to use them at the specific location where they are valid.

This concept of proximity-based service discovery overcomes the characteristic lack of a designated service infrastructure in ad hoc environments and is well suited for highly dynamic network topologies. The architecture of the Proximity Discovery Service is inherently distributed as it depends neither on centralized nor on intermediate components. A decentralized addressing scheme represents a key enabling mechanism to this inherently distributed architecture allowing mobile devices to recognize services of interest and to locally compute identifiers from service descriptions. The evaluation of the Proximity Discovery Service demonstrated that exploiting proximity to bound multicast-based service discovery limits the complexity of the underlying dynamic network topology without introducing overhead for maintaining discovery routes and overlay network structures. Such information typically needs to be updated more frequently with increasing migration speed of providers and clients causing the use of communication and computational resources to increase with the frequency of network topology changes. The evaluation also illustrated that exploiting proximity enables the Proximity Discovery Service to transparently select the appropriate protocol when disseminating advertisements.

Acknowledgements. The work described in this paper was partly supported by the Irish Higher Education Authority's Programme for Research in Third Level Institutions cycle 0 (1998-2001) and by the FET Programme of the Commission of the European Union under research contract IST-2000-26031 (CORTEX).

References

[1] M. Weiser, "Ubiquitous Computing," *IEEE Hot Topics*, vol. 26, pp. 71-72, 1993.

[2] P. Verissimo, V. Cahill, A. Casimiro, K. Cheverst, A. Friday, and J. Kaiser, "CORTEX: Towards Supporting Autonomous and Cooperating Sentient Entities," in *Proceedings of the European Wireless Conference*. Florence, Italy, 2002, pp. 595-601.

[3] E. Guttman, C. Perkins, J.Veizades, and M.Day, "Service Location Protocol, Version 2," IETF, 1999.

[4] K. Arnold, R. Scheifler, J. Waldo, B. O'Sullivan, A. Wollrath, B. O'Sullivan, and A. Wollrath, *Jini Specification*: Addison-Wesley Longman Publishing Co., Inc., 1999.

[5] Microsoft Corporation, "Universal Plug and Play: Background," 1999.

[6] R. Meier and V. Cahill, "Exploiting Proximity in Event-Based Middleware for Collaborative Mobile Applications," in *Proceedings of the 4th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS'03)*, *LNCS 2893*. Paris, France: Springer-Verlag, 2003, pp. 285-296.

[7] S. J. Mullender and P. M. B. Vitanyi, "Distributed Match-Making for Processes in Computer Networks (preliminary version)," in *Proceedings of the 4th Annual ACM Symposium on Principles of Distributed Computing*. Minaki, Ontario, Canada: ACM Press, 1985, pp. 261-271.

[8] Z. J. Haas and B. Liang, "Ad-Hoc Mobility Management with Randomized Database Groups," in *Proceedings of the IEEE International Conference on Communications (ICC 1999)*: IEEE Computer Society, 1999, pp. 1756-1762.

[9] Y. Xue, B. Li, and K. Nahrstedt, "A Scalable Location Management Scheme in Mobile Ad-Hoc Networks," in *Proceedings of the 26th Annual IEEE Conference on Local Computer Networks*: IEEE Computer Society, 2001, pp. 102-112.

[10] J. Li, J. Jannotti, D. S. J. D. Couto, D. R. Karger, and R. Morris, "A Scalable Location Service for Geographic Ad Hoc Routing," in *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*. oston, Massachusetts, USA: ACM Press, 2000, pp. 120-130.

[11] U. C. Kozat and L. Tassiulas, "Network Layer Support for Service Discovery in Mobile Ad Hoc Networks," in *Proceedings of IEEE INFOCOM*, vol. 23: IEEE Computer Society, 2003, pp. 1965-1975.

[12] R. Hermann, D. Husemann, M. Moser, M. Nidd, C. Rohner, and A. Schade, "DEAPspace: Transient Ad-Hoc Networking of Pervasive Devices," in *Proceedings of the 1st ACM International Symposium on Mobile Ad Hoc Networking & Computing*. Boston, Massachusetts, USA: IEEE Press, 2000, pp. 133-134.

[13] D. Chakraborty, A. Joshi, T. Finin, and Y. Yesha, "GSD: A Novel Group Based Service Discovery Protocol for MANETS," in *Proceedings of the 4th IEEE Conference on Mobile and Wireless Communications Networks (MWCN 2002)*. Stockholm, Sweden: IEEE Press, 2002.

[14] S. Helal, N. Desai, V. Verma, and C. Lee, "Konark -- A Service Discovery and Delivery Protocol for Ad-hoc Networks," in *Proceedings of the 3rd IEEE Conference on Wireless Communication Networks (WCNC)*. New Orleans, USA: IEEE Press, 2002.

[15] J. Orvalho, L. Figueiredo, and F. Boavida, "Evaluating Light-weight Reliable Multicast Protocol Extensions to the CORBA Event Service," in *Proceedings of the 3rd International Enterprise Distributed Object Computing Conference (EDOC'99)*. Mannheim, Germany, 1999.

[16] B. Preiss, *Data Structures and Algorithms with Object-Oriented Design Patterns in C++*: John Wiley & Sons, Inc., 1999.

[17] P. S. Wang, *C++ with Object-Oriented Programming*: PWS Publishing Company, 1994.

[18] R. Meier, "Event-Based Middleware for Collaborative Ad Hoc Applications," Department of Computer Science, University of Dublin, Trinity College, Ireland, Ph.D. Thesis September 2003.

[19] S.-J. Lee, W. Su, J. Hsu, M. Gerla, and R. Bagrodia, "A Performance Comparison Study of Ad Hoc Wireless Multicast Protocols," in *Proceedings of IEEE INFOCOM 2000*. Tel Aviv, Israel, 2000.

Enriching UDDI Information Model with an Integrated Service Profile

Natenapa Sriharee and Twittie Senivongse

Department of Computer Engineering, Chulalongkorn University,
Phyathai Road, Pathumwan, Bangkok 10330 Thailand
natenapa.s@student.chula.ac.th, twittie.s@chula.ac.th

Abstract. Service discovery is one key aspect in the enabling technologies for service-oriented systems including Web Services. Growing attention has been paid to the content of business and service descriptions to allow services to be discovered more flexibly and accurately. This paper presents an information model called an integrated service profile that covers aspects of Web Services, i.e. attribute-based profile and specification-based profiles. Attribute-based profile contains attributes that are described more easily with some values, while specification-based profiles represent attributes whose values are complex and then should be described as specifications. The specifications could describe Web Services in terms of their semantic structure, behaviour and rules, and composition. The paper proposes to represent these specifications by an ontology language, and hence they can be used further to discover Web Services semantically.

1 Introduction

The diversity of the format and content of service descriptions within a service-oriented environment has been problematic for service consumers when looking for available services. Standard UDDI registry for Web Services [1] attempts to standardise business and service descriptions through a set of business and service attributes. However, the attribute set is coarse and gives only preliminary information about the service providers and the offered Web Services. Generally a search is by matching of the name or category of Business Entities, Business Services, or tModels against the values specified in the query, such as *" I want to find service providers in the electronics appliance category"*. The search will return some information and the rest is left to the service consumer to browse the Web pages of those companies to make a selection. The search does not yet support a query that is also based on semantic or behavioural information such as *"I want to find an online electronics shop that sells desktop computers and is awarded Best Electronics Appliance Vendor from the Ministry of Commerce. The store should accept Amex credit card and deliver the computer that I have bought to my place (in Bangkok)"*.

It is assumed that service providers will do their best to please service consumers, and will try to advertise useful information as much as they can to facilitate the consumers as well as to get themselves discovered easily. Our work then started with the question "What should be in a service description to allow service consumers to query more conveniently and flexibly?" We started to survey on the service descriptions and their contents, and the result is an *integrated service profile*. This profile covers aspects of Web Services, i.e. *attribute-based profile* and *specification-based profiles*. Attribute-based profile contains attributes that are described more easily with some values, while specification-based profiles are associated with the attributes whose values are complex and then should be described as specifications. The specifications describe Web Services in terms of their *semantic structure, behaviour and rules, and composition*. This paper discusses the integrated service profile and reports some results of our study on defining and using the integrated profile.

Section 2 presents the attribute-based profile which is the result of the survey on the contents of service descriptions and Section 3 presents the specification-based profiles. Section 4 discusses the usage of the integrated service profile and some related work. The paper is concluded in Section 5.

2 Attribute-Based Profile: Survey on Web Services Descriptions

A survey was conducted to find in what way the description of a Web Service could be enriched. We gained some result from Web Services brokerage sites (such as www.capescience.com, www.webserviceoftheday.com, www.salcentral.com, www. xmethods.com), and we additionally looked further at the advertisements of software components on the Internet and at the literature on software components since software components and Web Services have similar characteristics (although there are different points) [2]. Considering both functional and psychological needs [3], the result of the survey is summarised in Fig. 1. (See [4] for details of the survey.) Most of the attributes here are optional, meaning that it is recommended to declare when applicable. The attributes that are shaded are not currently supported by standard UDDI but they can be imported by using specialised tModels that store the attribute values. Those that are not shaded are already supported by various parts of UDDI information model.

Most attributes in the attribute-based profile have values that are meaningful to service consumers (e.g. attribute Award) but those in the specification part of the profile represent more complex information that is hard to describe as attribute values. Therefore, each of these specification-based attributes may instead refer to (the URL of) the corresponding specification-based profile. We are interested in representing such profiles in a way that will enable a better service discovery, so we focus on ontology-based specifications (e.g. in OWL) for semantics-based discovery.

3 Specification-Based Profiles

Specification-based profiles will be derived from three upper ontologies (Fig. 2). The *structural upper ontology*, adapted from [5], represents semantic or knowledge structure of a Web Service. The *behavioural upper ontology*, corresponding to part of OWL-S service profile [6], represents service behaviour. The *rule upper ontology* represents a business rules policy or constraints that are put on the service behaviour.

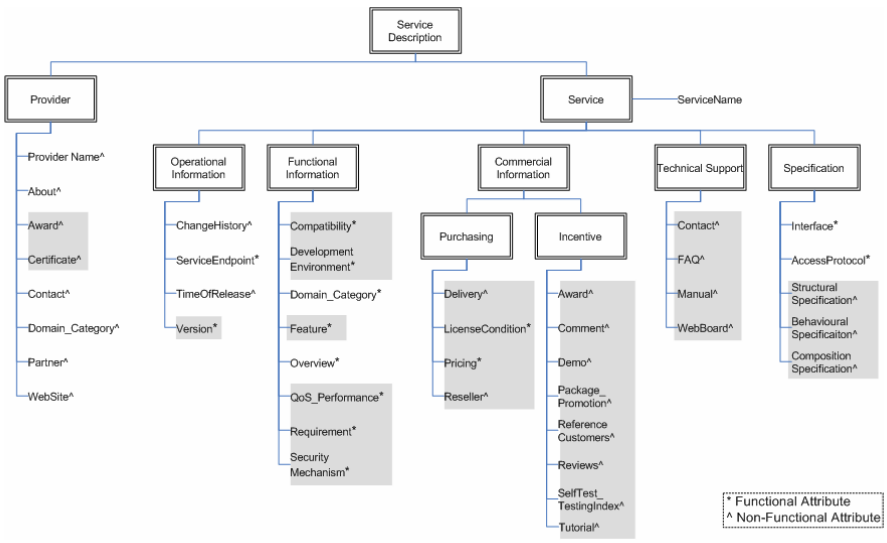


Fig. 1. Attribute-based profile

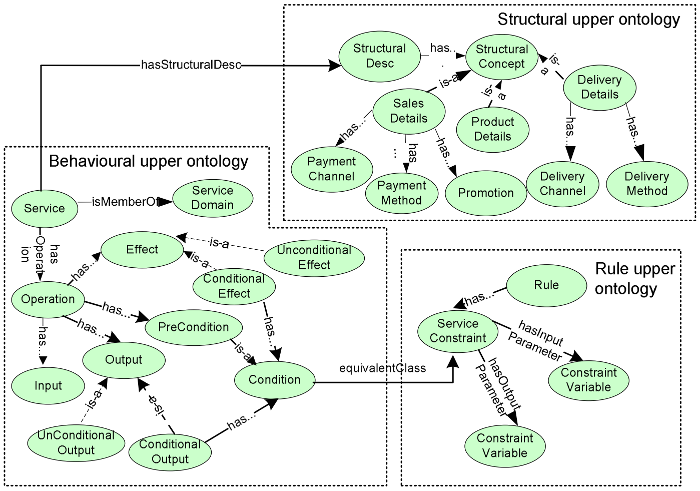
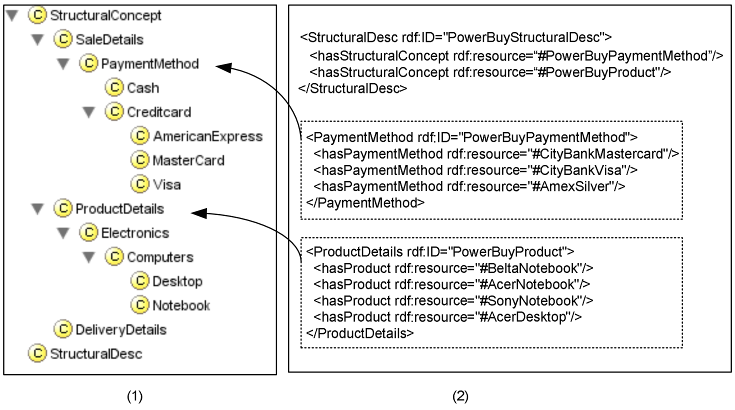


Fig. 2. Upper ontologies for specification-based profiles

From the upper ontologies, domain experts can derive corresponding domain-specific ontologies in order for the service providers to further derive their own profiles. Fig. 3(1) shows part of the structural ontology of the electronics appliance domain that is derived from the structural upper ontology. A service provider named PowerBuy then derives from this structural ontology to create its own structural profile in Fig. 3(2). Similarly, PowerBuy follows the behavioural ontology of the



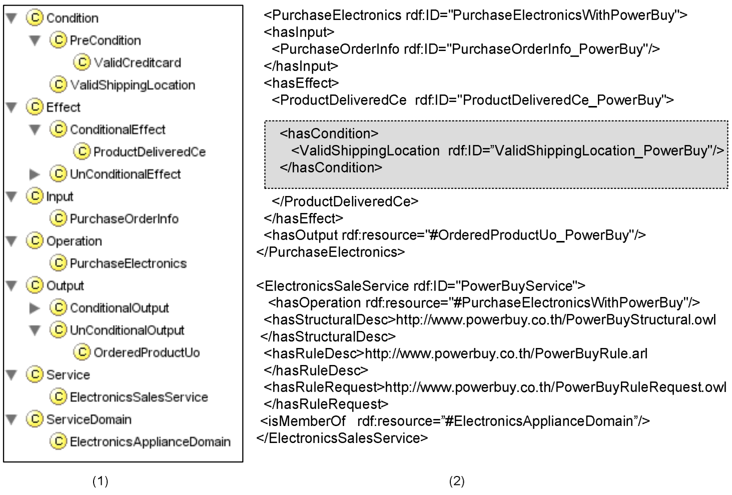
```
<StructuralDesc rdf:ID="PowerBuyStructuralDesc">
  <hasStructuralConcept rdf:resource="#PowerBuyPaymentMethod"/>
  <hasStructuralConcept rdf:resource="#PowerBuyProduct"/>
</StructuralDesc>

<PaymentMethod rdf:ID="PowerBuyPaymentMethod">
  <hasPaymentMethod rdf:resource="#CityBankMastercard"/>
  <hasPaymentMethod rdf:resource="#CityBankVisa"/>
  <hasPaymentMethod rdf:resource="#AmexSilver"/>
</PaymentMethod>

<ProductDetails rdf:ID="PowerBuyProduct">
  <hasProduct rdf:resource="#BeltaNotebook"/>
  <hasProduct rdf:resource="#AcerNotebook"/>
  <hasProduct rdf:resource="#SonyNotebook"/>
  <hasProduct rdf:resource="#AcerDesktop"/>
</ProductDetails>
```

Fig. 3. Structural specification (1) Part of structural ontology for electronics appliance domain (2) Part of structural profile of PowerBuyEshop service

electronics appliance domain in Fig. 4(1) to define its own behavioural profile (Fig. 4(2)). Note that PowerBuy has a condition ValidShippingLocation associated with its conditional effect – ProductDelivered. This means the product will be delivered only to some valid locations. Suppose that in this behavioural profile, ValidShippingLocation is defined as an equivalent class to the class ServiceShippingLocation which is in the rule ontology for the electronics appliance domain (Fig. 5(1)), PowerBuy can then create a rule profile for ServiceShippingLocation (Fig. 5(2)). The rule profile here refers to an associated rule definition, written in ABLE rule language (Fig. 5(3)). The rule definition says that the valid delivery locations are Bangkok and Rachaburi only.



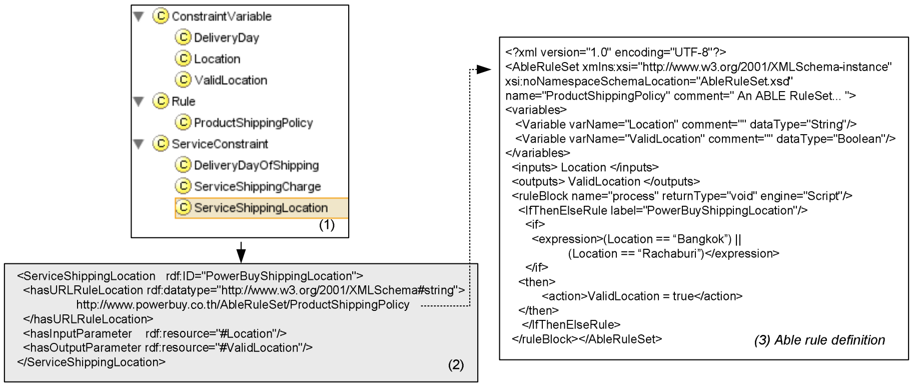
```
<PurchaseElectronics rdf:ID="PurchaseElectronicsWithPowerBuy">
<hasInput>
  <PurchaseOrderInfo rdf:ID="PurchaseOrderInfo_PowerBuy"/>
</hasInput>
<hasEffect>
  <ProductDeliveredCe  rdf:ID="ProductDeliveredCe_PowerBuy">

  <hasCondition>
    <ValidShippingLocation  rdf:ID="ValidShippingLocation_PowerBuy"/>
  </hasCondition>

  </ProductDeliveredCe>
</hasEffect>
<hasOutput rdf:resource="#OrderedProductUo_PowerBuy"/>
</PurchaseElectronics>

<ElectronicsSaleService rdf:ID="PowerBuyService">
  <hasOperation rdf:resource="#PurchaseElectronicsWithPowerBuy"/>
 <hasStructuralDesc>http://www.powerbuy.co.th/PowerBuyStructural.owl
</hasStructuralDesc>
 <hasRuleDesc>http://www.powerbuy.co.th/PowerBuyRule.arl
 </hasRuleDesc>
 <hasRuleRequest>http://www.powerbuy.co.th/PowerBuyRuleRequest.owl
 </hasRuleRequest>
<isMemberOf   rdf:resource="#ElectronicsApplianceDomain"/>
</ElectronicsSalesService>
```

Fig. 4. Behavioural specification (1) Behavioural ontology for electronics appliance domain (2) Behavioural profile of PowerBuyEshop service



```
<ServiceShippingLocation   rdf:ID="PowerBuyShippingLocation">
  <hasURLRuleLocation rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
        http://www.powerbuy.co.th/AbleRuleSet/ProductShippingPolicy
  </hasURLRuleLocation>
  <hasInputParameter    rdf:resource="#Location"/>
  <hasOutputParameter rdf:resource="#ValidLocation"/>
</ServiceShippingLocation>                                                    (2)
```

```
<?xml version="1.0" encoding="UTF-8"?>
<AbleRuleSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="AbleRuleSet.xsd"
name="ProductShippingPolicy" comment=" An ABLE RuleSet... ">
<variables>
  <Variable varName="Location" comment="" dataType="String"/>
  <Variable varName="ValidLocation" comment="" dataType="Boolean"/>
</variables>
 <inputs> Location </inputs>
 <outputs> ValidLocation </outputs>
 <ruleBlock name="process" returnType="void" engine="Script"/>
  <IfThenElseRule label="PowerBuyShippingLocation"/>
   <if>
     <expression>(Location == "Bangkok") ||
          (Location == "Rachaburi")</expression>
   </if>
  <then>
       <action>ValidLocation = true</action>
  </then>
   </IfThenElseRule>
 </ruleBlock></AbleRuleSet>                    (3) Able rule definition
```

Fig. 5. Rule specification (1) Rule ontology of electronics appliance domain (2) Rule profile of PowerBuyEshop service (3) Rule definition in ABLE rule language

4 Discussion

With the integrated service profile, a service consumer can submit a semantic query such as the one mentioned in Section 1. The attribute-based profiles of the providers will be searched to find ones that are in the electronics appliance domain and have received the Best Electronics Appliance Vendor award. Then the structural profiles of such providers will be checked if they sell desktop computers, and their behavioural profiles are checked if they accepts Amex card and have a delivery service. The qualified candidates will have their rule profiles checked further to see if they can make a delivery to the consumer's address. By representing the semantics of Web Service with an ontology language, the query can also benefit from ontology reasoning. For example, the providers who advertise that they sell a PC would match this query for a desktop shop, or if they advertise their effect as ProductDeliveredWithDeliveryCharge, they would also match the query for ProductDelivered effect. Details of the matching algorithm and the discovery architecture that supports this integrated service profiles and query can be found in [7].

Research work that attempts to enhance Web Services discovery mostly introduces semantics for service descriptions and focuses on only one aspect of the semantics. For example, UDDI version 4 is trying to incorporate an ontology-based taxonomy for the standard categories of Business Entity and Business Service [8]. This effort will allow UDDI to also return businesses or services of a specialised or generalised category. The work in [5] focuses on the knowledge about the service and corresponds to the use of our structural profile. The work in [9][10] focuses on searching by service behaviour and corresponds to the use of our behavioural profile, but they do not consider using precondition and effect as the query constraints whereas we do. We consider our integrated service profile closest to OWL-S effort (especially the OWL-S Service Profile) in that a building block for rich semantic service descriptions is developed. The behavioural profile may overlap with a part of OWL-S Service Profile but it is enhanced by the use of the rule profile. Also, our attribute-based profile is more extensive than the attributes in OWL-S Service Profile as it is a compilation from an empirical survey.

5 Conclusion

The proposed integrated service profile is in accordance with the Service-Oriented Model part of the Web Services Architecture [11] in which a Web Service is modelled to have information about the provider, the syntax and semantics of the service, choreography of the tasks within the service, and a business policy. It allows service consumers to compose more complex and comprehensive queries.

We are in the process of completing the prototype of the discovery framework that integrates with the standard UDDI. We are researching on how to determine the degree of matching and will continue to explore discovery by composition specification.

References

1. uddi.org: UDDI: Universal Description, Discovery and Integration of Web Services (online). http://www.uddi.org
2. Yang, J.: Web Service Componentization. Communications of the ACM Vol. 46 No. 10. October (2003) 35-40
3. O'Sullivan, et al.: What's in a Service? Towards Accurate Description of Non-Functional Service Properties. Distributed and Parallel Databases Vol. 12 (2002) 117-133
4. Teppaboot, C.: Attribute-Based Description Model for Distributed Services. Master Thesis, Dept. of Computer Engineering, Chulalongkorn University (2004)
5. Trastour, D. et al.: A Semantic Web Approach to Service Description for Matchmaking of Services. In: Proceedings of the International Semantic Web Working Symposium (SWWS'01) (2001)
6. Martin, D. et al.: Bringing Semantics to Web Services: The OWL-S Approach. In: Proceedings of 1st International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004), July (2004)
7. Sriharee, N.: Towards Semantic Discovery of Web Services Using an Integrated Service Profile. Tech. Report, Dept. of Computer Engineering, Chulalongkorn University (2005)
8. Paolucci, M., Sycara, K.: UDDI Spec TC V4 Proposal Semantic Search (online). (2004). http://www.oasis-open.org/committees/uddi-spec/doc/req/uddi-spec-tc-req029-semanticsearch-20040308.doc
9. Paolucci, M. et al.: Semantic Matching of Web Services Capabilities. Proceedings of the 1st International Semantic Web Conference (ISWC 2002), Sardinia (Italy), Lecture Notes in Computer Science, Vol. 2342. Springer Verlag (2002)
10. Sivashanmugan, K., Verma, K., Sheth, A., Miller, J.: Adding Semantics to Web Services Standards. Proceedings of the International Conference on Web Services (2003)
11. W3C: Web Services Architecture (online). (2004). http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/

Building a Configurable Publish/Subscribe Notification Service

C. Fiorentino[*], M. Cilia[**], L. Fiege, and A. Buchmann

Databases and Distributed Systems Group, Dept. of Computer Science
Technische Universität Darmstadt, Darmstadt, Germany
<lastname>@dvs1.informatik.tu-darmstadt.de
cfiorent@exa.unicen.edu.ar

Abstract. The convergence of technologies and information-driven applications require a middleware that supports data streams. This middleware needs to interpret, aggregate, filter and analyze streams of messages usually in a distributed environment. Publish/Subscribe middleware basically deals with some of these issues, but it is typically monolithic and includes only a subset of features. A problem arises when users want to find a middleware that completely fulfills their application requirements. Based on our experience, we propose a framework that allows the configuration/adaptation of a Pub/Sub solution based on a reusable and extensible set of components.

1 Introduction

New applications and the convergence of technologies, ranging from sensor networks to ubiquitous computing and from autonomic systems to event-driven supply chain management, require middleware platforms that support the handling of streams of data. In these cases, the underlying infrastructure needs to deal with the dissemination of data in a distributed environment from where it is produced to the destinations where consumers are located. Streaming data needs to be interpreted, aggregated, filtered, analyzed, reacted to and eventually disposed of. Publish/Subscribe (Pub/Sub) middleware basically deals with this problem.

Pub/Sub is founded on the principle that producers simply make information available and consumers place a standing request for information by issuing subscriptions. The pub/sub notification service (NS) is then responsible for making information flow from a producer (publisher) to one or more interested consumers (subscribers). Such a notification service provides asynchronous communication, it naturally decouples producers and consumers, makes them anonymous to each other, and allows a dynamic number of publishers and subscribers.

[*] Faculty of Sciences, UNICEN, Tandil, Argentina
[**] Also Faculty of Sciences, UNICEN, Tandil, Argentina

As you can imagine, there are commercial products (WebSphereMQ (formerly MQ-Series), TIB-Rendezvous, or pure JMS) that deal with some of these issues. In academia there are several projects that focus on one or the other issue of interest, like efficient dissemination, addressing models, message correlation, mobility, scalability, fault-tolerance, access control, data integration, security, privacy protection, transactions, caching, etc.

But specific requirements of a given application may need a combination of some of the previously mentioned aspects. For instance, a news ticker application basically requires topic-oriented addressing, efficient data dissemination and scalability, but does not include transaction support. On the other side, supply-chain management based on the AutoID Infrastructure requires content-based addressing, transactions, security, fault-tolerance, data integration and scalability, while other aspects like access control or caching are optional. Today no product does offer all features. Moreover, these products are often hardly extendible so that missing features cannot be added to fulfill all application requirements.

Based on our own experience in building notification services [16, 26, 25] and specific features for them [5, 10, 11, 4, 1, 3] and the experience of others [20, 21, 22, 23, 14, 7, 27], we develop a framework for building pub/sub notification services. It facilitates the combination of required features in a notification service, and as a result the middleware is 'shaped' towards application requirements. This offers us a testbed where different ideas can be experimented and proven together with other approaches.

In Section 2 we present the basic idea behind pub/sub notification services including an overview of related projects in this area, and we also include related work on service-based infrastructures. Section 3 enumerates the system requirements and sketches our proposal. In Section 4 we present our main design decisions including architecture, interfaces, and pre-defined components. Section 5 briefly describes the deployment strategy together with the runtime environment. Section 6 presents our conclusions and future work.

2 Background and Related Work

In a notification service there are different participants: *applications*, which basically produce and/or consume messages, and *brokers*, which help to disseminate messages across the network between producers and consumers. Each broker maintains a routing table that determines in which directions a message needs to be forwarded. This table needs to be maintained up-to-date with respect to active consumers and their subscriptions. We basically distinguish three types of brokers: *local brokers* constitute the clients' access point to the middleware and they are usually part of the communication library loaded into the clients. A local broker is connected to at most one border broker. *Border brokers* form the boundary of the distributed communication middleware and maintain connections to local brokers. *Inner brokers* are connected to other inner or border brokers and do not maintain any connections to the applications.

In recent years, academia and industry have concentrated on publish/subscribe mechanisms because they offer loosely coupled exchange of asynchronous notifications, facilitating extensibility and flexibility. These approaches have evolved from restricted channels to more flexible subscription mechanisms. For instance, subject-based addressing [17] define a uniform name space for messages and their destinations. Here, to every message a subject is attached in order to find matchings with subscriptions, also expressed with subjects.

To improve expressiveness of the subscription model the content-based approach was proposed where predicates on the content of a notification can be used for subscriptions. This approach is more flexible but requires a more complex infrastructure [2]. Many projects in this category (like Rebeca, Siena, JEDI, Gryphon) concentrate on scalability issues in wide-area networks and on efficient algorithms and techniques for matching and routing notifications to reduce network traffic [18, 15, 8]. Most of these approaches use simple Boolean expressions as subscription patterns and assume homogeneous name spaces.

More recently a new generation of publish/subscribe systems built on top of an overlay network has emerged. These systems mostly pursue wide-area scalability based on a topic-oriented addressing model. This is the case of Scribe [23] which is implemented on top of Pastry [22]. The mapping of topics onto multicast groups is done by simply hashing the topic name. Hermes [20] uses a similar approach but tries to get around the limitations of topic-based publish/subscribe by implementing a so-called "type and attribute based" publish/subscribe model. It extends the expressiveness of subscriptions and aims to allow multiple inheritance in event types. A content-based addressing on top of a dynamic peer-to-peer network was proposed in [26, 25] where the efficient routing of notifications takes advantage of the topology graph underneath. This work combines the high expressiveness of content-based subscriptions and the scalability and fault tolerance of a peer-to-peer system.

Most of the projects mentioned above are monolithic and they provide a static set of features. Few of them are in some sense extensible while the extensions are hard to develop. What is needed is the possibility to easily combine features in order to completely fulfill application requirements. This can be achieved if pub/sub notification services are built based on extensible set of components that rely on an appropriate architecture/infrastructure. The main requirements of such an infrastructure include: components' life-cycle management, remote and easy deployment, configurability, manageability, and monitoring capabilities. Component containers founded on the principles of inversion of control [13] fulfill some of these requirements.

Pico-Container is a small, light weight container but it does not provide components' management nor runtime configuration. DustDevil (a limited container implemented in Bamboo) is also light weight and has a nice desired communication structure but does not cover most of the previously mentioned requirements. JMX (Java Management eXtension) [24] is a little complex to use and heavy weight (not specially suited for small devices). OSGi (Open Services Gateway Initiative) [19] offers a container and management environment for managing service life-cycle. Services are bundles, JAR files, which contain classes and a configuration file. OSGi has a small footprint and is compliant with the J2ME specification. OSGi itself is built as configurable set services.

3 A Pub/Sub Notification Service Framework

3.1 System Requirements

Based on our own experience on building notification services [16, 26, 25, 5, 10, 11, 4, 1, 3] and the experience of others [20, 21, 22, 23, 14, 7, 27] we shortly enumerate here the main requirements that, from our perspective, need to be included in the resulting system.

As in every framework, *reusability* is an issue. Here, in particular, we want to reuse a predefined set of specific functionality, like serlialization of messages, routing algorithms, or topology-related strategies. But also *extensibility* in the sense of adding new features plays a fundamental role since optimizations and new routing strategies appear frequently. As notification services are usually distributed in a network of brokers, *easy* and *remote deployment* is a key factor. Moreover, in such a distributed NS *manageability* and *monitoring capabilities* are required in order to tune parameters or replace components if needed.

In most cases the user wants the resulting NS to be *efficient* and *scalable.* The possibility to allow the user to find a trade-off between these two factors is desired. Additionally, the possibility to build an *adaptable* NS where load- or fault-related adaptations can be managed. In many cases, the underlying infrastructure needs to run diverse and concurrent notification services just to fulfill the requirements of diverse applications, resulting on a unreasonable use of resources. This leads to an approach where the *rational use of resources* needs to be taken into account.

3.2 Proposed Approach

We want to provide a framework to build pub/sub notification services according to applications' requirements, selecting from a set of predefined NS characteristics like the underlying NS topology, routing strategies, message serialization and also defining other features like how to deal with failures, caching, security or access control. The resulting NS must be easily deployable. According to this, our goal is to offer an environment that supports the development of the application in question in the following three steps which are also sketched in Figure 1.

1. Based on the application requirements, the set of functions and mechanisms are selected from a pre-established component framework. These components can be simply extended by following clearly established interfaces. Selected components plus some Quality of Service (QoS) decisions are introduced through a configuration tool. After a generate operation, the desired building blocks are ready for the next phase.

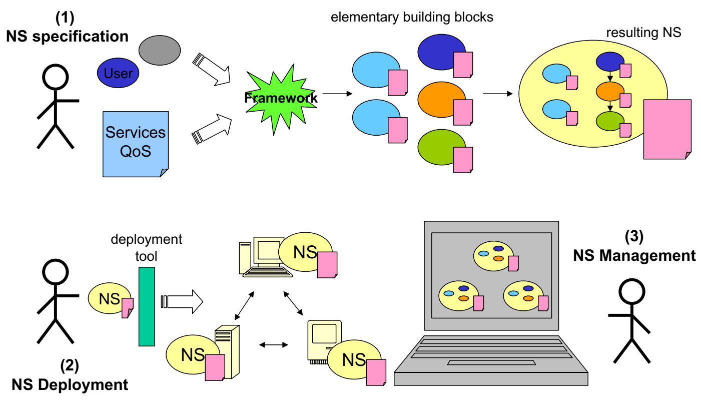


Fig. 1. Our Approach

2. The resulting NS is deployed by means of a deployment tool.
3. After the NS is deployed on the desired nodes, they can be monitored allowing calibration and tuning of the whole service where QoS parameters can be influenced.

4 Design Decisions

We have first analyzed and identified possible interactions among elementary building blocks of functionality based on a study that considered the projects mentioned in the related-work section [12]. We then grouped those blocks, what we call here *components*, according to their functionality and interaction pat-

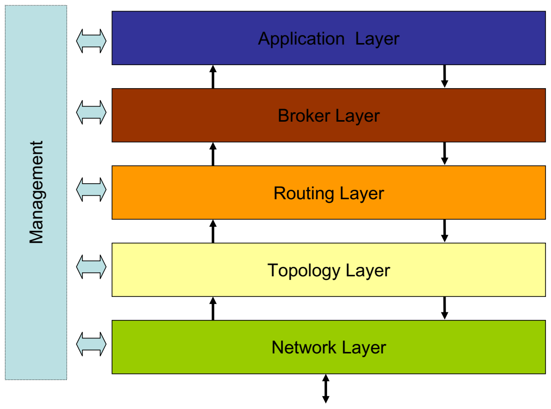


Fig. 2. High-level Architecture

terns. This analysis led us to a layering architecture. Figure 2 shows an overview of the architecture.

Based on the analyzed interaction patterns, we have defined the interfaces between layers. The communication among layers and components can be synchronous or asynchronous. Messages can be passed between layers as events by using the corresponding layer interface.

As it was mentioned before, a pub/sub NS relies on a distributed network of nodes that cooperate. The way this network is modelled has an enormous impact on the design of the infrastructure. For this reason, we assume an *Overlay* network which can be seen as the most generic case, where main characteristics of P2P (like self-organization and healing, robustness, load balancing, and scalability) are considered. Nevertheless, the static tree approach can still be modelled as a restricted overlay that does not offer such characteristics, since the nodes are simply static. Founded on [6] and our own experience we have defined a common overlay interface that condenses the kernel functionality of different overlay networks.

From the runtime perspective we focus our approach on a 'distributed' runtime environment that serves as a container for the resulting NS. This environment should support remote deployment and management, and also monitoring capabilities on the running NS.

The functionality to manage the whole system is orthogonal to the main stack of layers.

4.1 Interfaces

The design of a common set of interfaces between layers was crucial. They make possible to add and change functionality in every layer with minimal impact (if any) on others components.

Because of the nature of the system we are building (which is basically message passing), we can not restrict to synchronous interactions. Consequently, we provide two different kinds of interfaces[1]. Within the *asynchronous interfaces* we apply an event interaction. These interfaces are divided between pair-of-layers messages (also distinguishing between upwards and downwards) and connection-related messages. Figure 3 sketches the asynchronous interactions that may happen between layers.

Synchronous interfaces define a direct communication between layers and components. They are basically defined with the purpose to get or set state of components (in the same or neighbor layers), and also to get/set configuration parameters by the management functionality.

4.2 Components' Overview

In this section the responsibility of every layer is presented accompanied with a description of possible components at each corresponding layer.

[1] Due to space reasons we cannot present the complete definition of all interfaces.

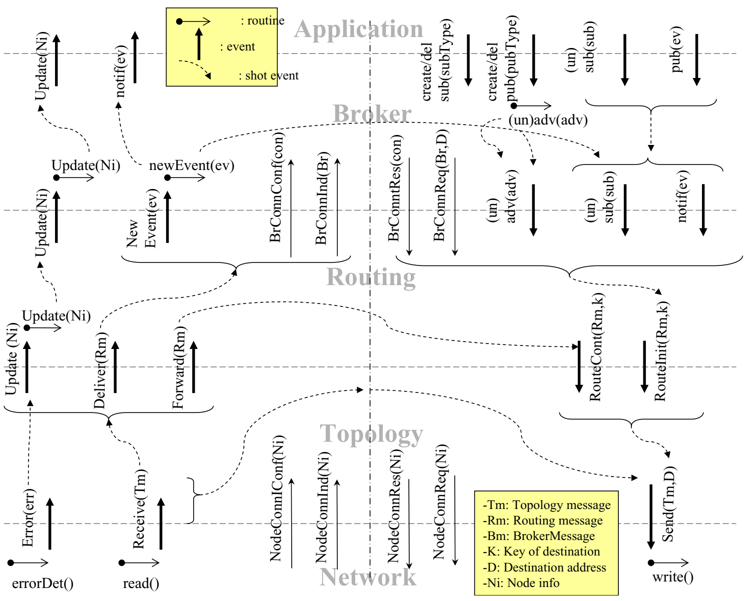


Fig. 3. Interfaces

Network Layer: Since this layer is at the bottom of the stack, its responsibility is oriented to the mediation with the physical network. On one side, messages coming from the topology layer need to be serialized and transmitted to a specific network address by using a determined protocol. On the other side, incoming messages from the protocol-side need to be de-serialized and passed to the upper layer. In this layer, different components can be used in order to apply diverse serialization or compression approaches and even different protocols.

Topology Layer: The main responsibility of this layer is to maintain the status information of the overlay up-to-date considering, for instance, dynamic changes like node joins and leaves, or errors. These situations must generate management events that need to be handled accordingly. Possible components within this layer are:

– **DHT**: is a specific kind of overlay network (bases on Distributed Hash Tables) that provides self-stability (dealing with failures and dynamic changes on the participant nodes), high scalability, good distributed object location in wide-area peer-to-peer applications, like Pastry and Bamboo.
– **Static tree**: is the simplest case of overlay networks. It is conformed as a static set of nodes, with fixed network physical addresses.
– **Channels**: may be defined as a dynamic set of paths, through which messages are transmitted. They can be thought as shortcuts within the network (reaching more directly different destinations). This improves performance and organization. This component can be combined with other components within the same layer.

Routing Layer: This layer performs the main message routing decisions. According to the addressing model, it must decide where to transmit specific events. Here, subscriptions, publications and advertisings are differentiated to decide when to change routing tables and transmit messages. The Routing Layer basically maintains message destinations according to the available subscription information. The destinations types may vary with respect to the selected service, for example they may be specific node addresses or just DHT keys. Within this layer, diverse components applying different routing strategies can be used, like the ones mentioned below:

– Traditional (Rebeca) routing algorithms [16], like Flooding, Simple Routing, Identity Routing, Covering, or Merging.
– P2P routing algorithms, like Bit Zipper, or the P2P approach presented in [11].
– The Ants approach [14], where the routing strategy is based on real ant behavior. This was simply traduced to a message routing algorithm that first floods the broker network, and then fills routing tables (according to some probabilities) to better route messages.
– The use of Scopes [9] which is an interesting routing strategy based on a selection criteria to restrict/group the set of message destinations.

Broker Layer: This broker basically offers the Pub/Sub functionality to the application layer. Additionally, it handles client and broker connections and message filtering. In general most events like subscriptions and notification reach this layer to be treated. Nevertheless, not all messages reach this layer due to

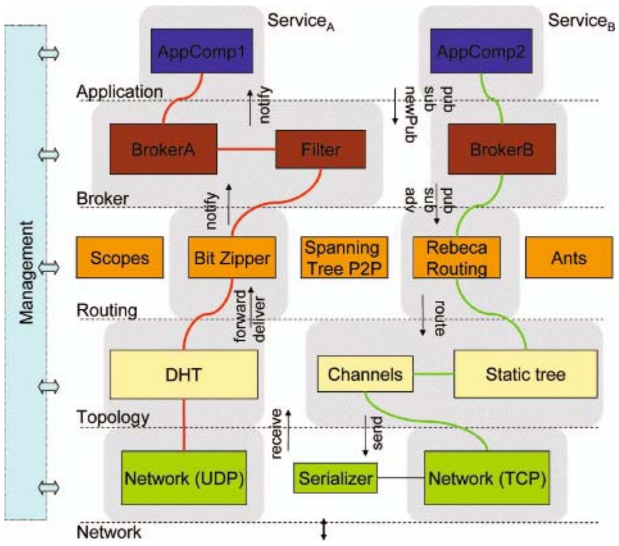


Fig. 4. Layers and components

shortcuts in the lower layers. Event correlation filters are handled at this layer. As mentioned before, brokers are classified on a) inner brokers, that manage the connections to other brokers and can apply correlation filters to the flow of messages b) border brokers, that handles connections with clients (which may offer, for instance, a store and forward functionality) and c) local broker, that basically manages the pub/sub interactions between the border broker and the client.

Application Layer: This layer is the one that includes application's logic related to NS usage. It accesses the NS by calling the pub/sub interface always on the broker layer. There may be at least two typical cases where this layer is used. The first one is in the case of P2P applications where the application is also a peer that runs broker functionality (see Figure 4). The other case is characterized by applications acting as a pure end consumer and/or producer of messages. They are not in charge of routing messages since they delegate this task to the corresponding border broker. In this particular case, the topology and routing layers are empty.

4.3 Services

Combining components from different layers is not trivial and there are obviously combinations that are inadequate. Dependencies among components are also represented within the framework. The combination of components crossing all layers is called here a *service*. Different services (or NS instances) can run concurrently within a single runtime environment. Moreover, and with the purpose of better using resources, a single component in a layer may serve various services within the same runtime environment.

In Figure 4 two services that bundle different components across layers is presented. $Service_A$ represents the BitZipper approach [25] relying on P2P infrastructure by using a UDP network connection component. On the other hand, $Service_B$ is characterized by a Rebeca routing strategy [16] relying on a tree topology combined with the use of channels and a traditional TCP network connection.

5 Deployment and Runtime

As can be clearly seen from the interfaces (Figure 3), they basically model three main roles within NS participants. The upward flow of data (left-side of the figure) basically represents *consumers of messages*, since messages are injected from the network layer and they leave the stack at the top (application layer). The downward flow (right-side of the figure) characterizes *producers of messages* since the flow begins at the top of the stack (application layer) and leaves it at the bottom. The third case basically combines the previous two cases and it is characterized by participants that act as brokers within the NS network. This is clearly visible in the figure by the upward flow (up to the broker layer) and the downward flow (up to the network layer) forming an inverted U. The picture also shows possible shortcuts within this flow representing routing optimizations. It must be noticed that NS participants can play just one of these roles or simply all roles simultaneously.

The identification of these flows within the layered architecture leads us to apply another idea for deployment purposes. We adopted a solution that is based on subscribing to components (instead of subscribing to certain messages) that offer certain features. This ends up with a pipeline of components, defining a straight communication between layers.

According to the specified service, a sequence of components can be built which basically represents the previously mentioned flow of messages. Depending on the roles established for participants, different sequences can be built for that purposes. At deployment-time, we build then pipelines of components (with limited control flow just to skip some of them if necessary) that basically process messages. This pipeline is ready to be deployed on a runtime environment.

We have selected OSGi as the runtime platform for our implementation. It offers most of our required functionalities: components life cycle management and remote configuration and deployment, it is light weight and easy to use.

Installing a whole NS is simple by relying on the OSGi platform. We obtain the desired NS as a file that contains all necessary components (in the form of service bundles). With this, the deployment of the NS can be performed (locally or remotely). After deployment, any component can be started and monitored. The system supports dynamic updates of new installed functionalities, gaining flexibility and runtime configurability.

6 Conclusions

We have presented in this paper a framework that allows system engineers to bundle components to satisfy a set of notification service requirements. The layered architecture of building blocks helps to understand, organize, and build notification service. This bundle is then deployed in our runtime environment that controls component life-cycle and offers monitoring capabilities that allow tuning and adaptation of the resulting NS. The presented solution covers pub/sub NSs that range from a light weight single purpose static NS to multi-broker, dynamic, remotely manageable NS. The decision of relying on an abstract overlay network simplifies the unification of different topology approaches.

Our solution for building pub/sub NSs can be used as a testbed for improvements on routing algorithms and other ongoing research projects. By having reusable components at hand, it is easy to pick the functionality you require for your experiments and also for probing these ideas under diverse NS constellations.

We have not (fully) automated the process of searching and selecting components from a repository. This is part of our ongoing work. Another pending task is a performance analysis comparing native pub/sub notification services like Rebeca, Hermes or BitZipper with their implementation based on our framework.

References

1. Jose Antollini, Mario Antollini, Pablo Guerrero, and Mariano Cilia. Extending rebeca to support Concept-Based addressing. In *In Proceedings of the Argentinean Symposium on Information Systems (ASIS'04)*, Cordoba, Argentina, September 2004.
2. A. Carzaniga, D. R. Rosenblum, and A. L. Wolf. Challenges for Distributed Event Services: Scalability vs. Expressiveness. In *Engineering Distributed Objects (EDO'99)*, Los Angeles, CA, May 1999.
3. M. Cilia, C. Bornhövd, and A. Buchmann. CREAM: An Infrastructure for Distributed, Heterogeneous Event-based Applications. volume 2172 of *LNCS*, Italy, November 2003. Springer.
4. Mariano Cilia, Mario Antollini, Christof Bornhvd, and Alejandro Buchmann. Dealing with heterogeneous data in pub/sub systems: The Concept-Based approach. In *International Workshop on Distributed Event-Based Systems (DEBS'04)*, Edinburgh, Scotland, May 2004.
5. Mariano Cilia, Ludger Fiege, Christian Haul, Andreas Zeidler, and Alejandro Buchmann. Looking into the past: Enhancing mobile publish/subscribe middleware. In *Proceedings of the 2nd International Workshop on Distributed Event-Based Systems (DEBS'03)*, San Diego, California, June 2003. ACM Press.
6. Frank Dabek, Ben Zhao, Peter Druschel, John Kubiatowicz, and Ion Stoica. Towards a common api for structured peer-to-peer overlays. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS03)*, Berkeley, CA, February 2003.
7. Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The Many Faces of Publish/Subscribe. *ACM Computing Surveys*, 35(2):114–131, 2003.
8. F. Fabret, F. Llirbat, J. Pereira, A. Jacobsen, K. Ross, and D. Shasha. Filtering Algorithms and Implementation for Very Fast Publish/Subscribe. pages 115–126, 2001.
9. L. Fiege, M. Mezini, G. Mühl, and A.P. Buchmann. Engineering Event-Based Systems with Scopes. In *Proc. of ECOOP'02*, volume 2374 of *LNCS*, 2002.
10. Ludger Fiege, Felix C. Grtner, Oliver Kasten, and Andreas Zeidler. Supporting mobility in Content-Based publish/subscribe middleware. In *ACM/IFIP/USENIX International Middleware Conference (Middleware 2003)*, pages 103–122, June 2003.
11. Ludger Fiege, Andreas Zeidler, Alejandro Buchmann, Roger Kilian-Kehr, and Gero Mhl. Security aspects in publish/subscribe systems. In *Third Intl. Workshop on Distributed Event-based Systems (DEBS'04)*, May 2004.
12. Cristian Fiorentino. Building a configurable notification service (under preparation). Master's thesis, Faculty of Sciences, UNICEN, Tandil, Argentina, April 2005.
13. Martin Fowler. Inversion of control containers and the dependency injection pattern. http://martinfowler.com/articles/injection.html, January 2004.
14. M. Gunes, U. Sorges, and I. Bouazzi. Ara – the ant-colony based routing algorithm for manets, 2002.
15. G. Mühl, L. Fiege, and A.P. Buchmann. Filter Similarities in Content-Based Publish/Subscribe Systems. In *Proc. of ARCS*, volume 2299 of *LNCS*, 2002.
16. Gero Mühl. *Large-Scale Content-Based Publish/Subscribe Systems.* PhD thesis, Darmstadt University of Technology, Germany, September 2002.

17. B. Oki, M. Pfluegl, A. Siegel, and D. Skeen. The Information Bus – An Architecture for Extensible Distributed Systems. In *Proceedings of the 14th Symposium on Operating Systems Principles (SIGOPS)*, pages 58–68, USA, December 1993.
18. Lukasz Opyrchal, Mark Astley, Joshua Auerbach, Guruduth Banavar, Robert Strom, and Daniel Sturman. Exploiting IP Multicast in Content-based Publish-Subscribe Systems. volume 1795 of *LNCS*, pages 185–207. Springer, 2000.
19. OSGi Alliance. The OSGi Service Platform. Technical report, July 2002.
20. Peter Pietzuch and Jean Bacon. Hermes: A distributed event-based middleware architecture. In J. Bacon, L. Fiege, R. Guerraoui, A. Jacobsen, and G. Mühl, editors, *In Proceedings of the 1st International Workshop on Distributed Event-Based Systems (DEBS'02)*, July 2002.
21. Sean Rhea, Dennis Geels, Timothy Roscoe, and John Kubiatowicz. Handling churn in a DHT. In *Proceedings of the 2004 USENIX Annual Technical Conference (USENIX '04)*, Boston, Massachusetts, June 2004.
22. Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218, 2001.
23. Antony I. T. Rowstron, Anne-Marie Kermarrec, Miguel Castro, and Peter Druschel. SCRIBE: The design of a large-scale event notification infrastructure. In *Networked Group Communication*, pages 30–43, 2001.
24. Sun Microsystems. Java Management Extensions. White paper, 1999.
25. Wesley W. Terpstra, Stefan Behnel, Ludger Fiege, Jussi Kangasharju, and Alejandro Buchmann. Bit zipper Rendezvous—Optimal data placement for general P2P queries. In *EDBT 04 Workshop on Peer-to-Peer Computing & DataBases*, 2004.
26. Wesley W. Terpstra, Stefan Behnel, Ludger Fiege, Andreas Zeidler, and Alejandro P. Buchmann. A Peer-to-Peer approach to Content-Based publish/subscribe. In *In Proceedings of the 2nd International Workshop on Distributed Event-Based Systems (DEBS'03)*, June 2003.
27. Matt Welsh, David E. Culler, and Eric A. Brewer. SEDA: An architecture for well-conditioned, scalable internet services. In *Symposium on Operating Systems Principles*, pages 230–243, 2001.

Protocol Reconfiguration Using Component-Based Design

Fotis Foukalas, Yiorgos Ntarladimas, Aristotelis Glentis, and Zachos Boufidis

Communications Network Laboratory,
Department of Informatics and Telecommunications,
University of Athens, Ilisia Campus 157 84,
Athens, Greece
{foukalas, yiorgos, arisg, boufidis}@di.uoa.gr
http://cnl.di.uoa.gr

Abstract. Previous modular protocol design and implementation allow a flexible configuration and reconfiguration of protocol layers or full protocol stacks. However, in our days, software engineering technologies introduce new methods for designing and specifying modular software. Such a technology is the component-based software technology. Using those techniques, a software system could be modular. This paper proposed a reconfigurable protocol design and specification approach as well as a protocol reconfiguration management/runtime model based on protocol components that represent distinct protocol functions, which in previous works have been designed and specified as modules. The following content could be considered as a suggestion for a UML profile for protocol components and protocol reconfiguration.

1 Introduction

This document is an attempt to introduce a UML profile for protocol components and how protocol reconfiguration can be achieved using component-based software design methods. The objective of this paper is the definition of the protocol component data model and the protocol component management model for reconfiguration purposes. The protocol component data model is the preclusive step that defines the stereotypes for modeling, specifying and implementing protocols as components. It is obvious that a protocol reconfiguration presupposes a reconfigurable way to design and specify communication protocols. Considering components as modular unit versus modules, we avoid the probability that a component will not include its manifestation in order to be replaced within its environment. The new component definition and specification, defined in UML 2.0, permits the component concept to be used to describe component designs or implementations, without losing the ability to describe deployment information [4].

In addition, the component-based design is one of the key enabling technologies for designing reconfigurable software. This design method supports the compositional adaptation approach of software systems that can be modified dynamically in terms of its structure and behavior [7]. The compositional adaptation of a software system is not just enabled for tuning software variables, like the parameter adaptation approach, but in contrast, enables the dynamic recomposition of the software during execution, for example to switch software components in and out of memory, or to add new behavior to deployed systems. Component-based design supports two types of composition the static composition and the dynamic composition. The second composition method is relied on late binding programming technique, which enables coupling of compatible components at runtime. It may be remarked that the dynamic composition is the only way in which a software engineer can add, remove or reconfigure components at runtime, although this method's flexibility can be constrained in order to ensure the system's integrity across adaptation.

The intention of this paper is to introduce the concept of protocol components and their reconfiguration that depend on the components and composite structures packages as well as the state machines package from infrastructure and superstructure specification of UML 2.0 [4][5]. In this direction, some steps for the component-based protocol design and specification have been identified [8]. Moreover, components for the physical and data link layer have been introduced by OMG for radio communications [9]. Related works also have introduced modular approaches for the design and implementation of protocol functions [1][10]. Moreover, an architecture that allows dividing protocol function into components using component-based software engineering and particularly the EJB Component Model has been addressed [11]. Considering the above-mentioned works and also the UML 2.0 specification, we are introducing the UML extension for protocol design and specification as well as reconfiguration management using the component classifier. A component-based design approach is the way to achieve a dynamic reconfiguration of communication protocols.

The rest of this document is organized as follows: In Section 2, we introduce the protocol component concept and definition and also a case study for implementing a protocol function as protocol component. Section 3 describes the protocol reconfiguration management model introducing the protocol component reconfiguration concept and also a user model for a MOF layer 0 protocol reconfiguration management model. In this section also is introduced the protocol manager as the entity for managing the protocol reconfiguration process. Section 4 discusses the protocol stack issues and how a stack can be constructed using protocol components and what are the requirements for a protocol stack reconfiguration. In addition, some future perspectives of our work are addressed. The last section of this paper is devoted to summary and conclusions.

2 Protocol Functions as Components

2.1 Protocol Component Definition

The idea for mapping protocol functions into modular software units has already introduced [1][3][10]. Protocol functions have already recognized as much as possible generic like the flow control, error control, segmentation/reassembly that represent well-known protocol functions. However, these generic protocol functions consist of more concrete protocol functions. For instance, error control protocol function consists of error correction, error detection and retransmission protocol functions as well as flow control consists of window-based and rate-based protocol functions [2]. Moreover, using different protocol mechanisms can specify protocol functions. Protocol mechanisms are like traditional protocol specifications: they have a unique name and they define the rules governing data transfer from one service user to its peer and handling of payload and control information [1]. On the other hand, in UML 2.0 the component concept has been clarified to be more definite and meaningful throughout the development life cycle. The UML 2.0 specification defines the component as a modular part of a system that encapsulates its contents and whose manifestation is replaceable within its environment [4][5]. Considering all the above features of communication protocols as well as the capabilities of component to be modulated, we have designed the following diagram (Fig. 1) that defines the protocol component concept.

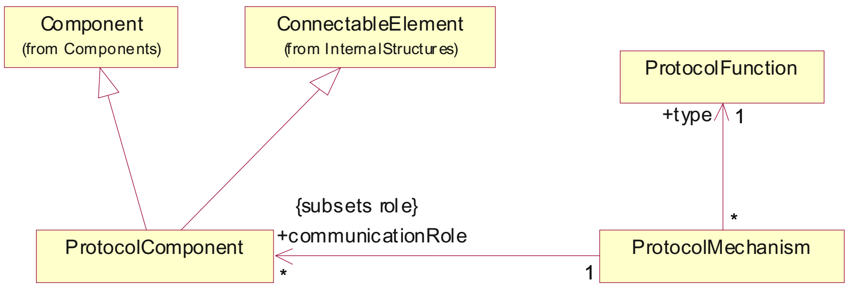


Fig. 1. Protocol Component definition

It has been recognized that a protocol mechanism consists of a sending and a receiving part, e.g., segmentation and reassembly, or the sending and receiving part of automatic repeat request (ARQ) [1]. *ProtocolMechanism* references protocol components, which can represent two definite roles the sender and/or receiver that instances play in this protocol mechanism. Therefore, the *ProtocolComponent* has a communication role in a protocol mechanism. Each *ProtocolComponent* represents the sender or the receiver role in a communication protocol mechanism. A protocol component can implement only one protocol mechanism and a protocol mechanism can be implemented by different protocol components. An example of protocol mechanism and protocol component association is given in the following paragraph.

Moreover, each protocol mechanism specifies only one protocol function and on the other hand each protocol function could be specified by several protocol mechanisms. The *type* association role defines the type of protocol function that the protocol mechanism specifies. Regarding the parents of *ProtocolComponent*, have been selected the Component from Components package and the ConnectableElement from InternalStructures package [4]. The parent Component indicates a protocol component as a kind of component since we have already envisioned a protocol component as a software unit that can be replaced within its environment. The parent ConnectableElement expresses a protocol component as a kind of connectable element that can be communicated with another instance through a network connection or something more simple as a pointer in case that the protocol component connects with another protocol component in the same node to form a protocol component stack. The following subsection gives an example of protocol mechanism specification using the extension in UML 2.0 for protocol components.

2.2 Protocol Component: A Case Study

In Fig.1 is illustrated the protocol components data model. The protocol components data model defines the basic stereotypes needed for the protocol design and specification using component-based software engineering methods and their associations with UML 2.0. In case that a protocol engineer needs to design and specify a protocol using protocol components, a question will be raised. What is the granularity of protocol functions and protocol mechanisms? It has been already recognized that the finer the granularity of protocol functions, the higher the number of different configurations and the higher the flexibility [1]. For instance, using a final granular approach could be specified an error detection protocol function that is a fine granular form of error control protocol function. In sequence, the error detection protocol function can be specified by a feedback error control mechanism, which finally can be implemented by a protocol component instance [2]. An example of the protocol component model for the fine granular protocol function error detection is depicted in the following figure (Fig. 2a). However, for a full specification of protocol mechanism a

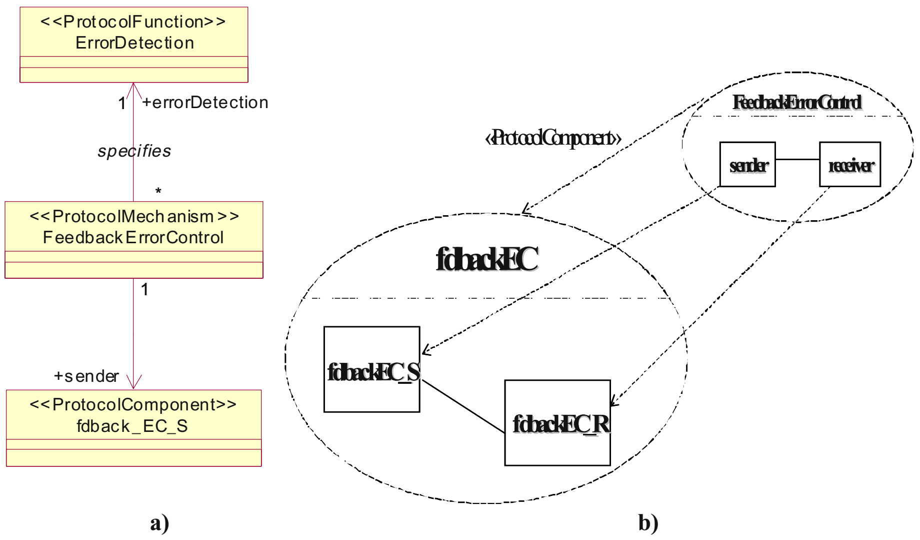


Fig. 2. a) Protocol Component Model for the error detection protocol function, - b) Protocol Mechanism Collaboration Diagram

collaboration diagram is needed in order to depict the communication role of the protocol component. The Fig. 2b depicts the collaboration diagram of error detection protocol mechanism and the roles of the protocol component instances. It is useful to denote that a protocol component could be developed implementing both the sending and receiving part [1]. In our case the *fdback_EC_S* protocol component implements only the sending part.

3 Protocol Components Management Model

3.1 Protocol Reconfiguration Management/Runtime Model

Protocol reconfiguration has been defined as the replacement of a component by another component that implements the same protocol mechanism [1]. According to 3GPP definition, reconfiguration is the rearrangement of the parts, hardware and/or software that make up the 3G network [15]. From the software engineering side, dynamic reconfiguration means the ability of code to be introduced at runtime [7]. Considering all the above definitions we define the dynamic reconfiguration as the process of the replacement/rearrangement of system parts which indeed must be performed at runtime. Taking into account this definition we are introducing the following diagram (Fig. 3) that represents the protocol reconfiguration management/runtime model. The management model is an active runtime entity that is dealing with (managing) data. The base data has been introduced in the previous section.

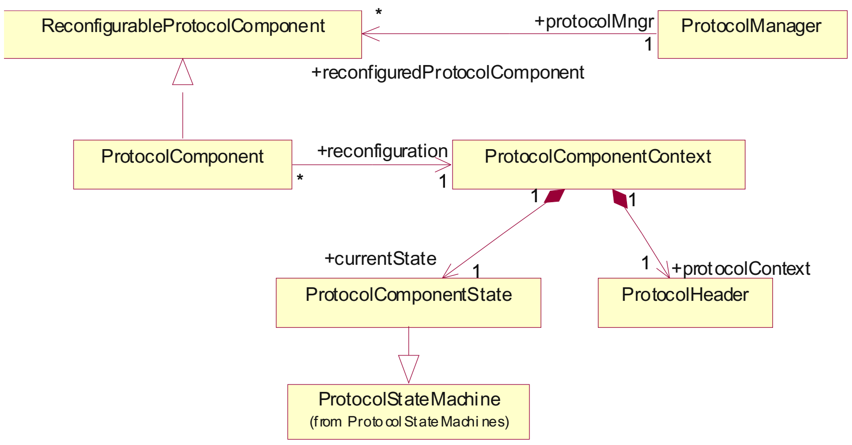


Fig. 3. Protocol Reconfiguration Management/Runtime Model

The context of the protocol component has been defined as the runtime entity that is going to be forwarded to the new component. The *ProtocolComponentContext* constitutes the reconfiguration of the *ProtocolComponent*. The association between the protocol component context and the component itself is 1:n since a protocol component context can be forwarded in numerous protocol components that implement the same protocol mechanism. The protocol component context is composed by protocol control information, actually, the header of the particular protocol data unit and the current state of the protocol component that is running. The *ProtocolComponent-State* is a kind of *ProtocolStateMachine* of StateMachines package [4]. Furthermore, in order to provide the ability for a protocol reconfiguration in a protocol software environment, we have introduced the interface *ReconfigurableProtocolComponent*, which provides all these operation for reconfiguration management. Through this interface a *ProtocolManager* can manage the reconfiguration process.

An example of protocol reconfiguration management user model, according to MOF layers, using the protocol components stereotypes and the protocol reconfiguration management model could be the following diagram (Fig. 4). This is based on dynamic protocol configuration and reconfiguration concept that has been introduced in document [1]. The *Ifdback_EC_S* is the unified interface of the protocol component provided for reconfiguration purposes. The operations of this interface enable to get and set the protocol context in order to reconfigure the protocol. There is also an *init* operation for the initialization of the protocol component with the appropriate protocol component context. Moreover, the interface for reconfiguration provides an operation for retrieving protocol component statistics for example the number of detected errors in a receiving CRC module. This last operation is provided for monitoring purposes and not for the realization of protocol component performance. At the moment, we are not going into details for protocol manager capabilities to select protocol components. Nevertheless, the following text encompasses a brief list of protocol manager capabilities in order to support the reconfiguration process.

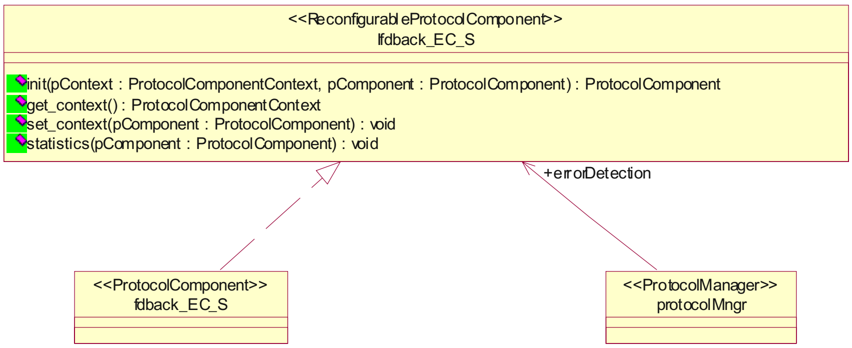

<<ReconfigurableProtocolComponent>>
Ifdback_EC_S
init(pContext : ProtocolComponentContext, pComponent : ProtocolComponent) : ProtocolComponent
get_context() : ProtocolComponentContext
set_context(pComponent : ProtocolComponent) : void
statistics(pComponent : ProtocolComponent) : void
+errorDetection
<<ProtocolComponent>>
fdback_EC_S
<<ProtocolManager>>
protocolMngr

Fig. 4. Reconfigurable Protocol Component

3.2 The Protocol Manager

The need for introducing the protocol manager has been raised since an entity for the following management procedures is necessary:

- A protocol manager must provide an efficient runtime environment for protocol configuration and reconfiguration.
- A protocol manager must link, initialize, and release protocol components, synchronize parallel components, and forward packets within stack of protocol components.
- A protocol manager must monitor the properties of protocol components
- A protocol manager must be enabled to realize protocol performance in order to decide the substitution between protocol components implemented the same protocol mechanism.
- A protocol manager must be enabled to map the requirements for protocol stack reconfiguration in the appropriate protocol selection and composition

However, the above-mentioned functionality is not an exhaustive list of protocol manager functionality. In addition, the protocol manager is envisioned as the entity that provides the time and storage management of the needs of a protocol component instance. The protocol manager will be also the entity that can parse the protocol component graph of a particular protocol stack. In the following section, we are discussing the requirements for a protocol component composition and selection and what they represent for a protocol reconfiguration.

4 Discussion

4.1 Requirements for Protocol Stack Reconfiguration

In most cases, the need for a reconfiguration process, in a modular software system, is being satisfied according to application requirements. In our case, a modular software system is the protocol stack of reconfigurable equipment [13]. A protocol stack reconfiguration has been considered as a reconfiguration process, which reconfigures a protocol or the stack as a whole, according to application requirements. The protocol stack is considered as protocol component graph that is composed by protocol components. Each protocol component implements a specific protocol mechanism required for the satisfaction of application requirements [1]. Application requirements represent the *required QoS* needed for running the appropriate application. The QoS requirements have to be mapped into protocol components properties for the selection of protocol components. The result of this mapping should be a reconfigured protocol stack providing the satisfaction of the application requirements.

However, a protocol stack reconfiguration requirements can be also surfaced from the properties offered by lower layers. For instance, a mobile device's protocol stack must be reconfigured in order to operate in a different radio technology. In such a case, the terminal can change its radio interface that includes properties. These properties represent the *offered QoS* and express the radio interface requirements. Some of the required radio-specific characteristics has already identified and indeed specified [12].

Therefore, considering the above mentioned reconfiguration scenarios we have recognized two dimension requirements; requirements that represent the required and offered QoS respectively (Fig.5).

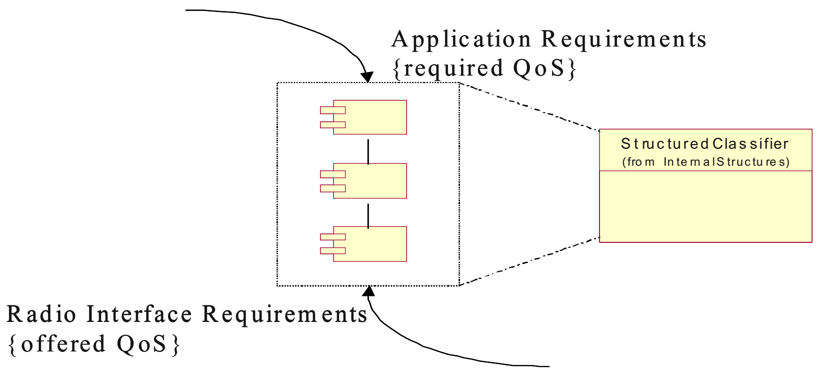


Fig. 5. Requirements for Protocol Stack reconfiguration

Using the definitions of several OMG's specification for components, we can define the requirement and property for protocol reconfiguration management. Requirement is the desired feature requested by component implementation [6]. Property represents a set of protocol component instances that are owned by a containing classifier instance [4]. In our case the containing classifier is the protocol stack that contains protocol components. The parent of protocol stack has been considered the structured classifier from the *InternalStructures* subpackage since the structured classifier provides the mechanisms for specifying structures of interconnected elements that are created within an instance of a containing classifier. Those interconnected elements are the protocol components that constitute the protocol stack as well as play the role of the protocol function that protocol components implement and a protocol mechanism specifies.

4.2 Future Work

In the previous subsection, we have envisioned and defined the protocol stack concept composed by protocol components. In this direction, we are considering the protocol component selection and composition according to radio interface and/or application requirements. The selection of the appropriate components according to requirements is enrolled in reconfigurable computing area. The requirements have to be mapped to the corresponding properties that encompass the criteria for the selection of protocol components in order a reconfiguration procedure takes place. We are working on this area and our future work will encompass the selection of protocol components according to radio interface requirements.

5 Summary – Conclusions

Many of the modular approaches for protocol software systems have been introduced the concept of module for protocol design, specification and implementation. However, the concept of software module does not include the deployment information.

Including deployment information a software system can be dynamically recomposed, installed and updated. All these procedures have been considered as the reconfiguration procedures of reconfigurable equipments [13]. We are envisioning a reconfigurable protocol stack that can be dynamically recomposed, installed and updated and for this reason we have introduced the protocol component concept.

References

1. Thomas Peter Plagemann "A Framework for Dynamic Protocol Configuration", PhD Thesis, Swiss Federal Institute of Technology Zurich, 1994.
2. Gerald J. Holzmann, "Design and Validation of Computer Protocols", Bell Laboratories, Prentice Hall, 1991.
3. L. Berlemann, A. Cassaigne, B. Walke, "Modular Link Layer Functions of a Generic Protocol Stack for Future Wireless Networks", to appear in Proc. of *Software Defined Radio-Technical Conference*, Phoenix USA, November 2004.
4. Object Management Group. UML 2.0 Superstructure Specification: Final Adopted Specification. http://www.omg.org/docs/ptc /03-08-02.pdf (August 2003).
5. Object Management Group. UML 2.0 Infrastructure Specification: Final Adopted Specification. http://www.omg.org/docs/ptc /03-09-15.pdf (December 2003).
6. Object Management Group. "Deployment and Configuration of Component-based Distributed Applications": Working Draft, http://www.omg.org/docs/ptc /04-05-15.pdf, 2002-2003.
7. Philip K. McKinley, Seyed Masoud Sadjadi, Eric P. Kasten, Betty H.C. Cheng , "Composing Adaptive Software", Computer, v.37 n.7, p56-64, July 2004.
8. A. Alonistioti, F. Foukalas, N. Houssos, "Reconfigurability management issues for the support of flexible service provision and reconfigurable protocols", Software Defined Radio Forum Technical Conference 2003 (SDR 2003), November 17-19, 2003, Orlando, Florida.
9. Object Management Group. "PIM and PSM for Software Radio Components": Final Adopted Specification , dtc/04-05-04.
10. C. Tschudin, "Flexible Protocol Stacks", Proc.ACM SIGCOMM '91, Zurich, Switcherland, 1991, pp. 197-204.
11. Matthias Jung and Ernst W. Biersack, "A Component-Based Architecture for Software Communication Systems", In Proceedings of IEEE ECBS, pages 36--44, Edinburgh, Scotland, April 2000.
12. Beyer, D.A.; Lewis, M.G., "A Packet Radio API", Page(s): 1261-1265 vol.3.
13. "End-to-End Reconfigurability", IP IST Project , http://e2r.motlabs.com/
14. Object Management Group. "CORBA Components ": Working Draft, ptc/02-08-03, September 2002.
15. 3GPP TS 32600, "Telecommunication Management; Configuration Management (CM); Concept and high-level requirements;(Release 6)".

A Secure and Efficient Communication Resume Protocol for Secure Wireless Networks

Kihong Kim[1], Jinkeun Hong[2], and Jongin Lim[3]

[1] National Security Research Institute,
161 Gajeong-dong, Yuseong-gu, Daejeon, 305-350, South Korea
hong0612@etri.re.kr

[2] Division of Information and Communication, Cheonan University,
115 Anse-dong, Cheonan-si, Chungnam, 330-740, South Korea
jkhong@cheonan.ac.kr

[3] Graduate School of Information Security, Korea University,
1, 5-Ka, Anam-dong, Sungbuk-ku, Seoul, 136-701, South Korea
jilim@korea.ac.kr

Abstract. There are important performance issues in secure wireless networks, such as mobility, power, bandwidth, and bit error rate (BER), that must be considered when designing a communication resume protocol. The efficiency of a secure session resume for a fast resume of secure communication is a key point in secure connection development. In this paper, a fast secure communication resume protocol using the initialization vector (IV) count for a secure wireless network is presented and evaluated against the efficiency of conventional resume protocols. Our proposed secure session resume protocol is found to achieve better performance, in terms of transmission traffic, consumed time, and BER, than conventional resume protocols with the same security capabilities.

1 Introduction

The wireless transport layer security (WTLS) provides privacy, authentication, and integrity in wireless application protocol (WAP) [1]. As the use of secure wireless networks becomes more widespread, the necessity of security for these networks is of increasing importance. However, in order to solve security issues in secure wireless networks, the efficiency of security services must be taken into account. From the point of view of wireless environmental characteristics, research on optimizing the security considerations of WTLS, such as low bandwidth, limited consumed power energy and memory processing capacity, and cryptography restrictions, has been presented [2] [3] [4] [5]. Secure session exchange key protocol and security in wireless communications have been researched by Mohamad Badra and Ahmed Serhrouchni [6], and by Mohammad Ghulam Rahman and Hideki Imai [7]. Hea Suk Jo and Hee Yong Youn [8] examined a synchronization protocol for authentication in wireless LANs, while Min Shiang Hwang et al. [9] proposed an enhanced authentication key exchange protocol. However, in terms of efficiency, the performance considerations for secure wireless networks, such as mobility, power, bandwidth, and BER, are very important. Of particular importance for a secure connection point is the efficiency of the secure session resume for the fast resume of secure communication.

In this paper, a protocol for fast secure session resume using IV count in secure wireless networks is presented and its performance is evaluated against that of conventional resume protocols. Results shows that the proposed protocol achieves better performance in terms of transmission traffic, consumed time, and BER than conventional resume protocols with the same security capabilities. Of particular note is that the proposed protocol reduces consumed time by up to 60.6 %, compared with conventional protocols in a wireless network environment.

The remainder of this paper is organized as follows. In the next section, detailed descriptions of the conventional full handshaking and session resume protocol are given. In section 3, the proposed secure session resume protocol is illustrated. Some performance considerations are presented in section 4, and concluding remarks are provided in section 5.

2 Conventional Key Handshaking Protocol in WTLS

The WTLS protocol determines the session key handshaking mechanism for secure services and transactions in secure wireless networks, and consists of the following phases: the handshaking phase, the change cipher spec phase, and the record protocol phase (RP) [3] [4] [5]. In the handshaking phase, all the key techniques and security parameters, such as protocol version, cryptographic algorithms, and the method of authentication, are established between the client and the server. After the key handshaking phase is complete, the change cipher spec phase is initiated. The change cipher spec phase handles the changing of the cipher. Through the change cipher spec phase, both the client and the server send the finished message, which is protected by a RP data unit that is applied by the negotiated security suites [6] [7]. The RP phase is a layered protocol phase that accepts raw data to be transmitted from the upper layer protocols. RP compresses and encrypts the data to ensure data integrity and authentication. It is also responsible for decrypting and decompressing data it receives and verifying that it has not been altered. In terms of secure wireless networks, WTLS requires fewer cryptographic computations, fewer resources, and less processing time than the secure sockets layer (SSL) protocol [1] [3].

Secure communication necessitates the encryption of communication channels. To achieve this, a key handshaking protocol allows two or more users to share a key or an IV. A conventional key handshaking protocol is illustrated in Fig. 1. The client sends a client hello message that includes information such as the version, session ID, acceptable cipher suites, and client random. When the server receives the client hello message, it responds with a hello message to the client and it also sends its certificate, key exchange, certificate request, and server hello done message. After receiving the server hello done message, the client responds by authenticating itself and sending its certificate.

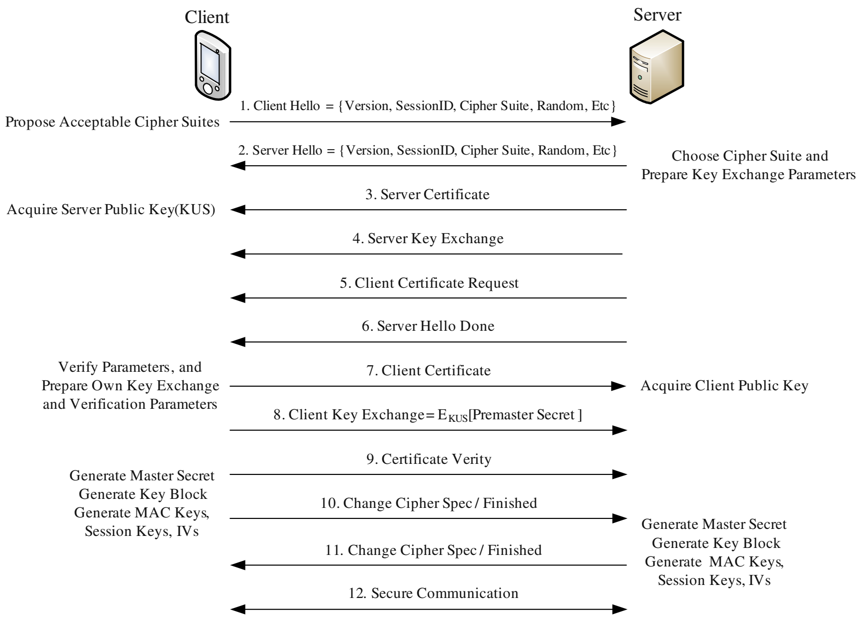

Client
Server
Propose Acceptable Cipher Suites
1. Client Hello = {Version, SessionID, Cipher Suite, Random, Etc}
2. Server Hello = {Version, SessionID, Cipher Suite, Random, Etc}
Choose Cipher Suite and Prepare Key Exchange Parameters
Acquire Server Public Key(KUS)
3. Server Certificate
4. Server Key Exchange
5. Client Certificate Request
6. Server Hello Done
Verify Parameters, and Prepare Own Key Exchange and Verification Parameters
7. Client Certificate
Acquire Client Public Key
8. Client Key Exchange= E_KUS[Premaster Secret]
9. Certificate Verity
Generate Master Secret Generate Key Block Generate MAC Keys, Session Keys, IVs
10. Change Cipher Spec / Finished
11. Change Cipher Spec / Finished
Generate Master Secret Generate Key Block Generate MAC Keys, Session Keys, IVs
12. Secure Communication

Fig. 1. Full handshaking process in WTLS protocol

Then, the client generates the premaster secret and sends its encryption data $E_{KUS}[Premaster\ Secret]$ encrypted with the server's public key to the server. The premaster secret is used to generate a master secret that is shared between the client and the server.

The client then generates the master secret using the premaster secret, client random, and server random. It also generates a sufficiently long key block using the master secret, client random, and server random [1]. The generated key block is hashed into a sequence of secure bytes, which are assigned to the message authentication code (MAC) keys, session keys, and IVs. This is represented as follows in Eq. (1) and Eq. (2).

$$\begin{aligned} Master\ \ Secret\ =\ &Pseudo\ Random\ Function(Premaster\ Secret,\\ &''Master\ Secret'',\ Client\ Random\ +\ Server\ Random) \end{aligned} \tag{1}$$

$$\begin{aligned} Key\ \ Block\ =\ &Pseudo\ Random\ Function(Master\ Secret,\\ &''Key\ Expansion'',\ Client\ Random\ +\ Server\ Random) \end{aligned} \tag{2}$$

The client sends a change cipher spec message and proceeds directly to the finished message in order to verify that the key exchange and authentication process were successful. The server also generates MAC secrets, session keys, and IVs using the key block. Then it sends the finished message to the client. Finally, secure communication over the secure connection is established using session keys and IVs.

2.1 Protocol for Secure Session Resume Using Premaster Secret

After completion of the conventional full handshaking protocol shown in Fig. 1, a secure communication between client and server is established. However, data frame loss occurs because of bit slips, channel loss, reflection, and diffraction in the communication channel. If a data frame is lost, the output of the decryptor will be unintelligible for the receiver and a session resume will be required. The aim of the session resume is to ensure that the encryptor and decryptor have the same internal state at a certain time. An internal state different from all previous sessions has to be chosen to prevent the reuse of session keys or IVs [10] [11] [12].

To overcome the problems caused by these data frame losses, resume protocols for secure communication have been suggested. Such protocols can be achieved by one of two methods: 1) premaster secret regeneration and retransmission, which results in a new master secret and new key block, or 2) random regeneration and retransmission, in which random is only used to change the key block in each secure session resume.

Fig. 2 shows a protocol for a secure session resume using premaster secret regeneration and retransmission. In this protocol, a new premaster secret is generated and sent in each session resume, and thus it results in the generation of a new master secret and new key block. Therefore, new session keys and new IVs are generated for every session resume. However, since a new premaster secret is generated and sent in each secure communication resume, this method has disadvantages such as a large computation load, time delay (including channel delay), and BER. This protocol is executed as follows. First, secure communication between the client and server is performed for time Δt, and then data

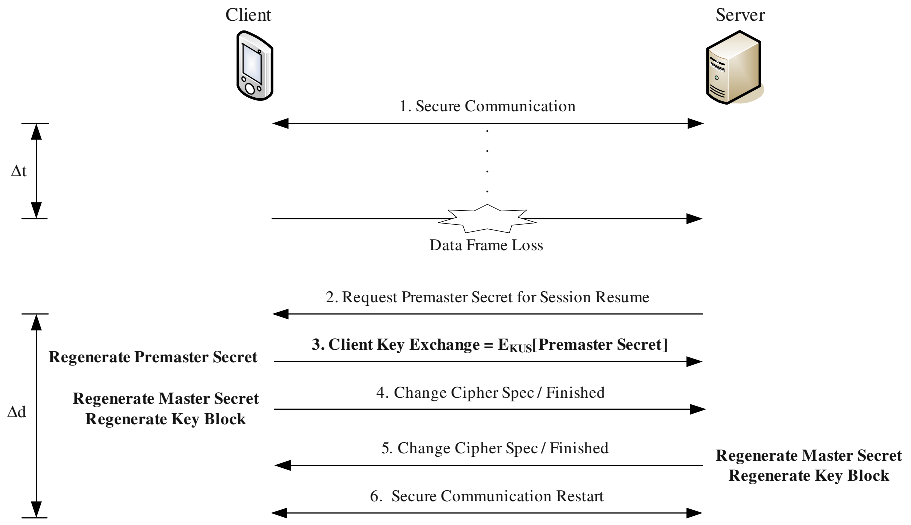

Client
Server
1. Secure Communication
Δt
Data Frame Loss
2. Request Premaster Secret for Session Resume
Regenerate Premaster Secret
3. Client Key Exchange = E_KUS[Premaster Secret]
4. Change Cipher Spec / Finished
Regenerate Master Secret
Regenerate Key Block
Δd
5. Change Cipher Spec / Finished
Regenerate Master Secret
Regenerate Key Block
6. Secure Communication Restart

Fig. 2. Conventional protocol for secure session resume using premaster secret

frame loss occurs. After the server realizes the data frame loss, it requests a new premaster secret for session resume. The client generates a new premaster secret and sends $E_{KUS}[Premaster\ Secret]$. The client then generates a new master secret using the new premaster secret and the original random cached in the initial hello message stage, and generates a new key block using the new master secret and original random. Thus, the result is the generation of new session keys and new IVs.

$$New\ Master\ Secret = Pseudo\ Random\ Function(New\ Premaster\ Secret, "Master\ Secret", Original\ Client\ Random + Original\ Server\ Random) \tag{3}$$

$$New\ Key\ Block = Pseudo\ Random\ Function(New\ Master\ Secret, "Key\ Expansion", Original\ Client\ Random + Original\ Server\ Random) \tag{4}$$

The client then sends the finished message to the server. The server generates a new master secret and a new key block, and then also sends the finished message to the client. After the session resume time Δd shown in Fig. 2, secure communication is reinitiated.

2.2 Protocol for Secure Session Resume Using Random Value

On the other hand, the protocol for a secure communication resume using random regeneration and retransmission is shown in Fig. 3. In this protocol, a new random is generated and sent in each secure session resume, which results in the generation of a new key block in each session resume. As with premaster secret regeneration and retransmission, this protocol also suffers from time delay, including channel delay, and a large BER.

Secure communication is performed for time Δt, and then data frame loss occurs. After realizing the data frame loss, the server requests a new random for session resume. The client generates a new random and includes it in a hello message. After the server receives the hello message from the client, it sends its own hello message that includes its new random. The server also generates a new key block using the new random and cached original master secret, and then generates new session keys and new IVs. This means that a resumed session will use the same master secret as the previous one. Note that, although the same master secret is used, new random values are exchanged in the secure communication resume. These new randoms are taken into account in the new key block generation, which means that each secure connection starts up with different key materials: new session keys and new IVs.

$$New\ Key\ Block = Pseudo\ Random\ Function(Original\ Master\ Secret, "Key\ Expansion", New\ Client\ Random + New\ Server\ Random) \tag{5}$$

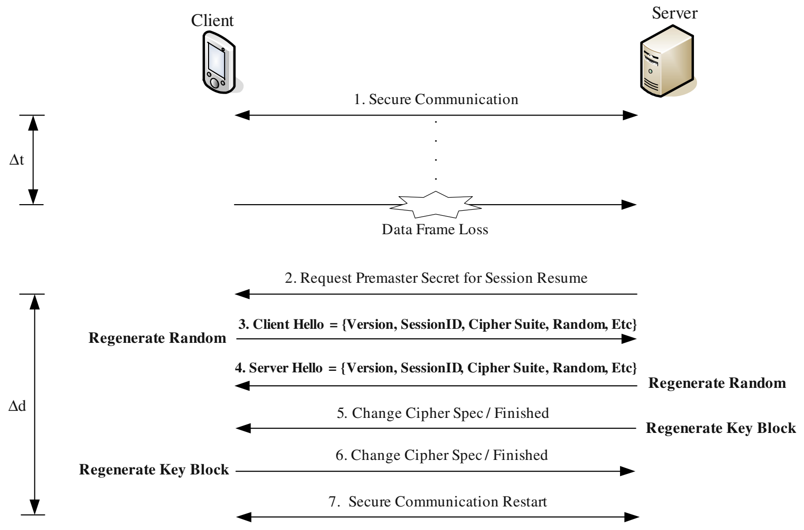

Client
Server
1. Secure Communication
Δt
Data Frame Loss
2. Request Premaster Secret for Session Resume
3. Client Hello = {Version, SessionID, Cipher Suite, Random, Etc}
Regenerate Random
4. Server Hello = {Version, SessionID, Cipher Suite, Random, Etc}
Regenerate Random
Δd
5. Change Cipher Spec / Finished
Regenerate Key Block
6. Change Cipher Spec / Finished
Regenerate Key Block
7. Secure Communication Restart

Fig. 3. Conventional protocol for secure session resume using random value

Finally, the server sends the finished message to the client. The server generates a new key block, resulting in new session keys and new IVs, and then it also sends the finished message to the client. After session resume time Δd shown in Fig. 3, secure communication is reinitiated.

3 Proposed Secure Communication Resume Protocol Using IV Count

3.1 Protocol for Proposed Secure Session Resume Using IV Count

To overcome the problems inherent in conventional secure communication resume protocols and to reinitiate secure communication much faster than they allow, we propose a new, efficient, and secure communication resume protocol that uses an IV count value.

Fig. 4 shows the proposed secure communication resume protocol, in which a count value of IV is sent to generate the new IVs in each secure session. After realizing the data frame loss, the server requests a new count value of IV for session resume. The client sends a new count value IV and generates new IVs using the count value. That is, the count value is used to generate new message protection materials, which means that each secure connection starts up with different IVs. Therefore, a resumed session will use the same session keys as the previous session. Note that, although the same session keys are used, new IVs are used in the secure communication resume. The client sends the change cipher spec and finished message to the server. The server generates new IVs using the received count value, and then sends the change cipher spec and finished message

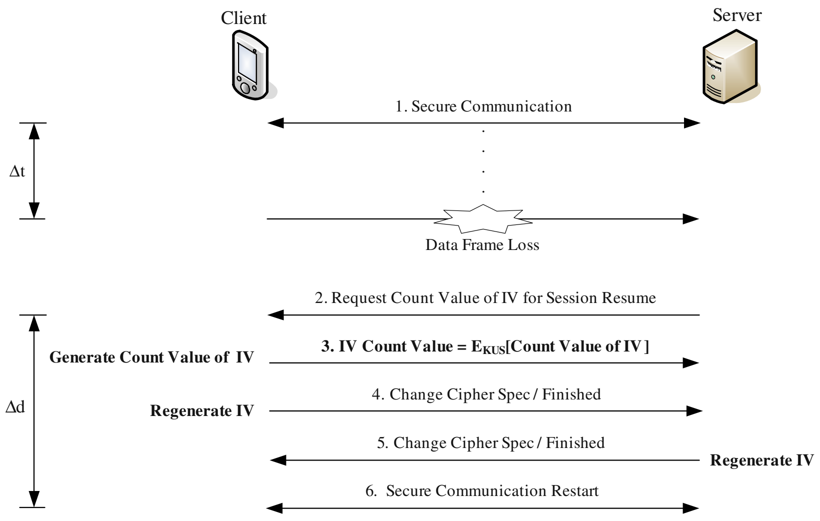

Client
Server
1. Secure Communication
Δt
Data Frame Loss
2. Request Count Value of IV for Session Resume
Generate Count Value of IV
3. IV Count Value = E_KUS[Count Value of IV]
Δd
Regenerate IV
4. Change Cipher Spec / Finished
5. Change Cipher Spec / Finished
Regenerate IV
6. Secure Communication Restart

Fig. 4. Proposed protocol for secure session resume using IV count

to the client. The client and server finally have the new IVs after session resume time Δd as shown in Fig. 4 and as represented in Eq. (6).

$$\alpha = \alpha_0 + \nu, \quad 1 \leq \nu \leq 2^{IV\ Size} - 1 \tag{6}$$

Here, α is the value of IV in each session and α_0 represents the value of the original IV. ν is a count value in each session resume and is increased by a value of one for every session resume.

3.2 Security Analysis

Security problems regarding attacks against the WAP WTLS were surveyed by Markku Juhani Saarinen [5], and it has been found that many of the changes that were made by the WAP Forum have led to increased security problems [1]. In this paper, to determine the key refresh period for secure session resume, the key refresh concept, which is referred by the WAP forum, was used and the condition of low bound was derived to avoid collisions from the birthday paradox [13].

An internal state different from all previous sessions has to be chosen, to prevent the reuse of session keys or IVs. If two n bit ciphertexts, C_i and C_j, are arbitrarily chosen from ciphertext block $C = (C_1, C_2, \ldots, C_M)$, and provided that the input plaintext $P = (P_1, P_2, \ldots, P_M)$ in the cipher block chaining (CBC) mode [13] are equal, the following Eq. (7) is given.

$$C_i = E_K(P_i \oplus C_{i-1}) = E_K(P_j \oplus C_{j-1}) = C_j \tag{7}$$
$$P_i \oplus C_{i-1} = P_j \oplus C_{j-1}, \quad P_i \oplus P_j = C_{i-1} \oplus C_{j-1}$$

Also, if the two ciphertexts, C_i and C_j, in cipher feed back (CFB) mode [13] are equal and if the two key stream blocks, O_i and O_j, in output feed back (OFB) mode [13] are also equal, these are represented as follows in Eq. (8).

$$\begin{aligned} E_K(C_i) \;=\; P_{i+1} \oplus C_{i+1} &= P_{j+1} \oplus C_{j+1} \;=\; E_K(C_j) \\ P_{i+1} \oplus P_{j+1} &= C_{i+1} \oplus C_{j+1} \\ O_i \;=\; E_K(O_{i-1}) \;=\; P_i \oplus C_i &= P_j \oplus C_j \;=\; E_K(O_{j-1}) \;=\; O_j \\ P_i \oplus P_j &= C_i \oplus C_j \end{aligned} \tag{8}$$

In other words, we acquire the information of plaintexts from the known ciphertexts. Therefore, the new keys are calculated and updated after a proper period to overcome the problematic plaintexts information issue. This key refresh period is computed using the birthday paradox. The number of pairs generated with ciphertext block $C = (C_1, C_2, \ldots, C_M)$ is $M(M-1)/2$, and the probability of at least one coincidence is shown in Eq. (9).

$$\begin{aligned} P &= 1 - \left(1 - \frac{1}{2^n}\right)^{\frac{M(M-1)}{2}} = 1 - \left(1 - \frac{1}{2^n}\right)^{-2^n \times \frac{M(M-1)}{2^{n+1}}} \\ &\approx 1 - e^{\frac{M(M-1)}{2^{n+1}}} \approx \frac{M(M-1)}{2^{n+1}} \end{aligned} \tag{9}$$

where n bit is the block size used in the block cipher. If M is about $2^{n/2}$, then at least one coincidence is found. By the birthday paradox, for strong collision resistance and a well-designed block cipher function with n bit input block size, it must hold that finding any pair $(x, y) \ni f(x) = f(y)$, takes $2^{n/2}$ trials. In the birthday bound, this means that even a perfect n bit block cipher function will start to exhibit collisions when the number of inputs nears the birthday bound $2^{n/2}$. Then, if coincidence exists, the problem of the plaintexts information issue occurs. Consequently, a new key is generated and updated before encrypting the $2^{n/4}$ input plaintext blocks. For example, in the case of DES, the maximum key refresh period is $2^{16}(2^{64/4})$ plaintext blocks.

$$T_{Key\ Refresh} = 2^{n/4} \tag{10}$$

Table 1. Result of collision probability and input/output block size

Block Size (n)	Number of Input Block (M)	Probability (P)
64 bits	2^{8}	1.77×10^{-15}
	$2^{16}(2^{64/4}$, key refresh period)	1.16×10^{-10}
	2^{32}	4.99×10^{-1}
128 bits	2^{16}	6.31×10^{-30}
	$2^{32}(2^{128/4}$, key kefresh keriod)	2.71×10^{-20}
	2^{64}	4.99×10^{-1}

Table 1 shows the relation of collision probability and block size in the bound of the birthday paradox.

On the other hand, IV resets after 2^{IVSize}. The probability of IV reset within the $2^{n/4}$ key refresh period is as small as the IV size is large, while the probability of its reset is as large as the IV size is small. For instance, if an IV size is 8 bytes, it resets after 2^{64}. This means that 8 bytes IV do not reset within the 2^{16} key refresh period, namely, $2^{64/4}$ input plaintext blocks in 64 bits block size. Therefore, if data frame loss occurs within the key refresh period, and if then a secure session resume is required, we have only to generate and update a new IV for every session resume instead of a new key generation. In addition, we have only to generate and update new keys at the time of key refreshing.

4 Performance Consideration

In this paper, to prove the efficiency of the proposed protocol, we compared and analyzed the transmission message size and the consumed time for session resume of our proposed protocol with conventional protocols. The performance of the proposed session resume protocol has also been evaluated in terms of BER.

Table 2 shows a comparison of the transmission procedure and message sizes according to each protocol for secure session resume: CLT is the client, SVR is the server, the Change Cipher Spec/Finished message is CCS/F, V is WTLS version, and SID is session ID, R is random, and SI is a security association such as key exchange suit, cipher suit, compression method, etc.

As shown in Table 2, in the premaster secret protocol, the transmission messages consist of the premaster secret, client change cipher spec/finished message, and server change cipher spec/finished message. The transmission size of these messages is about 46 bytes. In the case of random protocol, the transmission messages are about 86 bytes in size and are composed of the client hello mes-

Table 2. Comparison of Δd and transmission message size according to each protocol

Protocol for Resume	Steps	Δd	Transmission Message Size
Premaster Secret	3	Premaster Secret	20 bytes
		CLT CCS/F	13 bytes
		SVR CCS/F	13 bytes
Random	4	[V, R, SID, SI] CLT Hello	30 bytes
		[V, R, SID, SI] SVR Hello	30 bytes
		CLT CCS/F	13 bytes
		SVR CCS/F	13 bytes
Proposed	3	IV Count Value	8 bytes
		CLT CCS/F	13 bytes
		SVR CCS/F	13 bytes

Table 3. Consumed time to reopen secure session

Protocol for Resume		2G at 100bps	2G at 9.6Kbps	3G at 14.4Kbps	3G at 384Kbps
Premaster Secret	T1	3.7 sec	38 ms	25 ms	1 ms
	TC1	7.4 sec	76 ms	51 ms	2 ms
	TC3	22.1 sec	230 ms	153 ms	5 ms
Random	T1	6.9 sec	70 ms	47 ms	1.7 ms
	TC1	13.7 sec	140 ms	95 ms	3.5 ms
	TC3	41.3 sec	430 ms	280 ms	10.7 ms
Proposed	T1	2.7 sec	28 ms	18 ms	0.7 ms
	TC1	5.4 sec	56 ms	30 ms	1 ms
	TC3	16.3 sec	170 ms	113 ms	4.2 ms

sage, server hello message, client change cipher spec/finished message, and server change cipher spec/finished message. However, in the proposed protocol, the transmission messages are composed only of the count value of IV, client change cipher spec/finished message, and server change cipher spec/finished message and their size is about 34 bytes. This Table shows that our proposed session resume protocol allows the establishment of secure sessions in an economic way, as it has fewer transmission message flows and smaller sizes than either the premaster secret or the random protocol. This can be particularly advantageous in wireless networks where the radio bandwidth is bottlenecked.

To evaluate the efficiency of secure session resume protocol, the consume time needed for each protocol must also be considered. The results of the consumed time for session resume are shown in Table 3. For 2G and 3G in a bearer service environment, each protocol is serviced by 100bps (9.6Kbps), i.e., a minimum(maximum) bandwidth environment of 2G, and 14.4Kbps (384Kbps), i.e., a minimum(maximum) bandwidth environment of 3G. Here, T1 are the transmission bits at each 1 iteration, TC1 are the transmission bits at each 1 iteration with 50 % redundancy channel coding, and TC3 are the transmission bits at 3 iterations with 50 % redundancy channel coding.

In the TC1 environment, 2G at 100bps, if the protocol used for session resume is the premaster secret protocol, the consumed time for session resume is about 7.4 sec. In the case of random protocol, the consumed time is even higher about 13.7 sec. In the proposed protocol, however, the consumed time is only about 5.4 sec, proving that this protocol provides a faster secure session resume than the other resume protocols.

Table 4 shows results of BER in 3G according to the number of session resumes using the consumed time in Table 3. When computing the BER for 1 session resume number per hour in the TC1 environment, the BERs in each protocol are provided: 5.03×10^{-7} in premaster secret protocol, 9.72×10^{-7} in random protocol, and 2.78×10^{-7} in the proposed protocol. This means that the proposed protocol reduces BER by over 45 % when compared with the premaster secret protocol, and by about 72 % when compared with the random protocol.

Table 4. BER according to the number of session resumes (per hour)

Protocol for Resume		♯1	♯2	♯4	♯6	♯8
Premaster Secret	T1	2.78×10^{-7}	5.56×10^{-7}	1.11×10^{-6}	1.67×10^{-6}	2.22×10^{-6}
	TC1	5.03×10^{-7}	1.11×10^{-6}	2.22×10^{-6}	3.33×10^{-6}	4.44×10^{-6}
	TC3	1.26×10^{-6}	2.78×10^{-6}	5.55×10^{-6}	8.33×10^{-6}	1.11×10^{-5}
Random	T1	4.72×10^{-7}	9.44×10^{-7}	1.89×10^{-6}	2.83×10^{-6}	3.78×10^{-6}
	TC1	9.72×10^{-7}	1.94×10^{-6}	3.89×10^{-6}	5.83×10^{-6}	7.78×10^{-6}
	TC3	2.97×10^{-6}	5.94×10^{-6}	1.19×10^{-5}	1.78×10^{-5}	2.38×10^{-5}
Proposed	T1	1.94×10^{-7}	3.89×10^{-7}	7.78×10^{-7}	1.16×10^{-6}	1.56×10^{-6}
	TC1	2.78×10^{-7}	5.56×10^{-7}	1.11×10^{-6}	1.67×10^{-6}	2.22×10^{-6}
	TC3	1.17×10^{-6}	2.33×10^{-6}	4.67×10^{-6}	7.00×10^{-6}	9.33×10^{-6}

We can also see that BERs increases as the number of session resumes increases, and also that when the proposed protocol is used, the BERs are small than when the premaster secret or random protocol is used.

Fig. 5 shows a BER at the T1 environment shown in Table 4, demonstrating that the BERs from the proposed protocol are smaller than those from the premaster secret and random protocols.

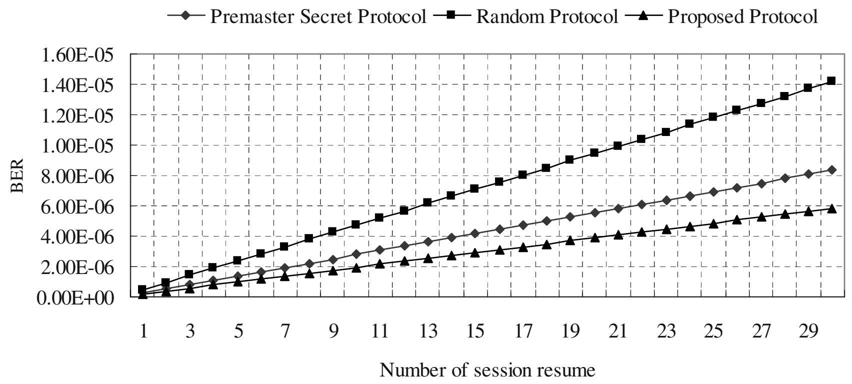


Fig. 5. BER at T1 environment in 384Kbps according to the number of session resumes

5 Conclusion

Most security research in secure wireless networks is focused on secured routing and transmitting in the network. However, because of the security issues in secure wireless networks, we suggest that the efficiency of security services is also an important issue. In this paper, a fast and secure communication resume protocol using IV count for wireless networks is presented and evaluated against the efficiency of conventional resume protocols. During the secure session resume phases, we manage to reduce transferring traffic and thus also reduce the bandwidth on wireless networks. Moreover, our enhanced proposed protocol is able to reduce the consumed time or cryptographic load and the computations in order to reopen secure sessions quickly.

Therefore, this proposed secure session resume protocol provides a fast resume of secure communications, while having the same security capabilities as other protocols and reducing the transferring traffic, consumed time, and BER in a WTLS protocol environment. In particular, this protocol reduces consumed time by up to 60.6 % when compared with conventional protocols.

References

1. WAP Forum. Wireless Transport Layer Security Spec. http://www.wapforum.org.
2. Sami Jormalainen, Jouni Laine. Security in the WTLS. http://www.tml.hut.fi/Opinnot/Tik-110.501/1999/papers/wtls.htm.
3. Ramesh Karri, Piyush Mishra. Optimizing the Energy Consumed by Secure Wireless Sessions-WTLS Case Study. *Mobile Networks and Applications*, No. 8, Kluwer Academic Publishers, 2003.
4. Philip Mikal. WTLS : The Good and Bad of WAP Security. http://www.advisor.com/Articles.nsf/aid/MIKAP001, 2001.
5. Markku-Juhani Saarinen. Attacks against the WAP WTLS Protocol. http://www.freeprotocols.org/harm0fWap/wtls.pdf, 1999.
6. Mohmad Badra et al.. A New Secure Session Exchange Key Protocol for Wireless Communication. *IEEE International Symposium on Personal, Indoor and Mobile Radio Communication*, 2003.
7. Mohammad Ghulam Rahman, Hideki Imai. Security in Wireless Communications. *Wireless Personal Communications*, No. 22, Kluwer Academic Publishers, 2002.
8. Hea Suk Jo, Hee Young Youn. A New Synchronization Protocol for Authentication in Wireless LAN Environment. *ICCSA'04*, LNCS publishers, 2002.
9. Min Shiang Hwang et al.. On the Security of an Enhanced Authentication Key Exchange Protocol. *AINA'04*, LNCS publishers, 2004.
10. Joan Daemen, Rene Govaerts, Joos Vandewalle. Resynchronization Weakness in Synchronous Stream Ciphers. *Pre-proceeding of EUROCRYPT'93*, 1993.
11. Randall K. Nichols, Panos C. Lekks. Wireless Security - Models, Threats, and Solutions. *McGraw-Hill Telecom*, 2002.
12. Amoroso E. Fundamentals of Computer Security Technology. *PTR Prentice Hall*, Englewood Cliffs, New Jersey, 1993.
13. Bruce Schneier. *Applied Cryptography*, 2nd ed, John Wiley and Sons Inc., 1996.

An Architecture for Collaborative Scenarios Applying a Common BPMN-Repository

Thomas Theling[1], Jörg Zwicker[1], Peter Loos[1], and Dominik Vanderhaeghen[2]

[1] Johannes-Gutenberg University Mainz, Information Systems and Management, 55099 Mainz, Germany
{Theling, Zwicker, Loos}@isym.bwl.uni-mainz.de
http://www.isym.bwl.uni-mainz.de

[2] Institute for Information Systems at the German Research Center for Artificial Intelligence (DFKI), 66123 Saarbruecken, Germany
vanderhaeghen@iwi.uni-sb.de
http://www.iwi.uni-sb.de/

Abstract. Collaborative business scenarios require a flexible integration of enterprises. To manage inter-organizational business processes, existing concepts for business process management (BPM) need to be adopted and extended. This paper presents a conceptual architecture for cross-enterprise processes' planning, implementation, and controlling. Core component of the architecture is a distributed repository managing all required data and information, which especially obtains a process-oriented view on collaboration networks. Cross-organizational processes need close coordination among networking partners. This is achieved through the integration of business process models. Thus we further propose a procedure model of an XML-based model transformation, which reduces complexity of the modeling-task and enables model integration on a conceptual level.

1 Collaborative Business Process Management Field of Research

Innovation pressure arises in interaction relations like supply chains and in networks with complementary core competence partners. Thus the concept of borderless enterprises [1] has been discussed for years and the collaborative production of goods and services has been established. The opening of organizational borders is no longer regarded as a necessary evil, but rather as a chance with strategic importance. Therefore inter-organizational applications and the transfer of business process management aims into collaborative IT systems are considered.

An architecture for managing cooperative and collaborative business processes requires functions which support all phases of collaborations [2]. Existing architecture concepts are discussed e. g. by Gronau, Scheer, van der Aalst, or Wild et al. [3-6]. They stress various single aspects and tasks within collaborations. Workflow-functionalities [7] for the IT-supported execution of processes was developed, while EAI-functionalities [8] ensure data integration and process integration. Business process management methods were steadily developed and enhanced. Web Service technologies [9] are proposed to facilitate process management based on application components. Service-oriented Architectures (SOA) [10] are established for composing a process-oriented, flexible and reactive information system architecture.

To exchange data between architecture components, data and especially business process models have to be stored in unique data formats. Transformation of models is addressed in the field of software development with the use of approaches like the model-driven architecture (MDA) [11]. Furthermore existing business process knowledge should be reused in collaborations. Notation-specific XML representations like the Event-driven Process Chain Markup Language (EPML) [12] focus on business process model transformations. Techniques as Extensible Stylesheet Language Transformations (XSLT) [13], support the automated transformation of XML-based models between different, heterogeneous formats in general. Our approach designs a framework for the transformation into a collaboration-wide business process description format.

This contribution depicts the concept of an Architecture for Collaborative Scenarios and to exemplify a procedure model enabling syntactic business process model interoperability. The paper unfolds as follows: After this introduction, we address the Architecture for Collaborative Scenarios in the following chapter. Our approach considers organizational aspects and assimilates existing concepts of workflow management and BPM frameworks [4, 5, 7]. Further on, to fulfill the aforementioned requirement of a collaboration-wide unique syntax of business process models BPMN is suggested in chapter 3 for storaging of these models within the introduced architecture. For solving the problem of heterogeneity in business process modeling, a generic, adaptable procedure model to gain syntactic model interoperability is developed – showing the steps for the transformation in a collaboration-wide BPMN syntax. Thus, we provide the preparing methods for the integration of process models in collaborations in a holistic approach. Finally, conclusions of our approach are discussed. Also, we point to some further research directions.

2 Architecture for Dynamic Collaborative Scenarios

Dealing with collaborations and information systems, the term dynamic concerns the change of people, the change of technologies and the change of processes [14]. Further dynamic aspects are e. g. the time lag between IT-investments and IT-payoffs [15]. Main requirement resulting from dynamic factors is flexibility. In order to reconfigure or to resolve a collaboration, to integrate a new partner into collaborations or to plan new collaborative strategies, indicators have to be defined and depicted. Further on, changing processes require an ad-hoc-execution and adaption of business processes and structures. This implies a flexible integration of various operational IT systems.

Further demands for an integrative architecture are e.g. graphical, multi-perspective modeling while supporting different modeling methods, integrated collaboration management and controlling, integrated process execution based upon the stored business process models, and a concept for integration of new network partners. At the same time existing solutions have to be considered and assimilated.

Based on these requirements the Architecture for Collaborative Scenarios has been developed (cf. figure 1). Basic information for the process execution is visualized in business process models, output models, and organization models. These are created by modeling tools and stored in a distributed repository. Depending on process- and

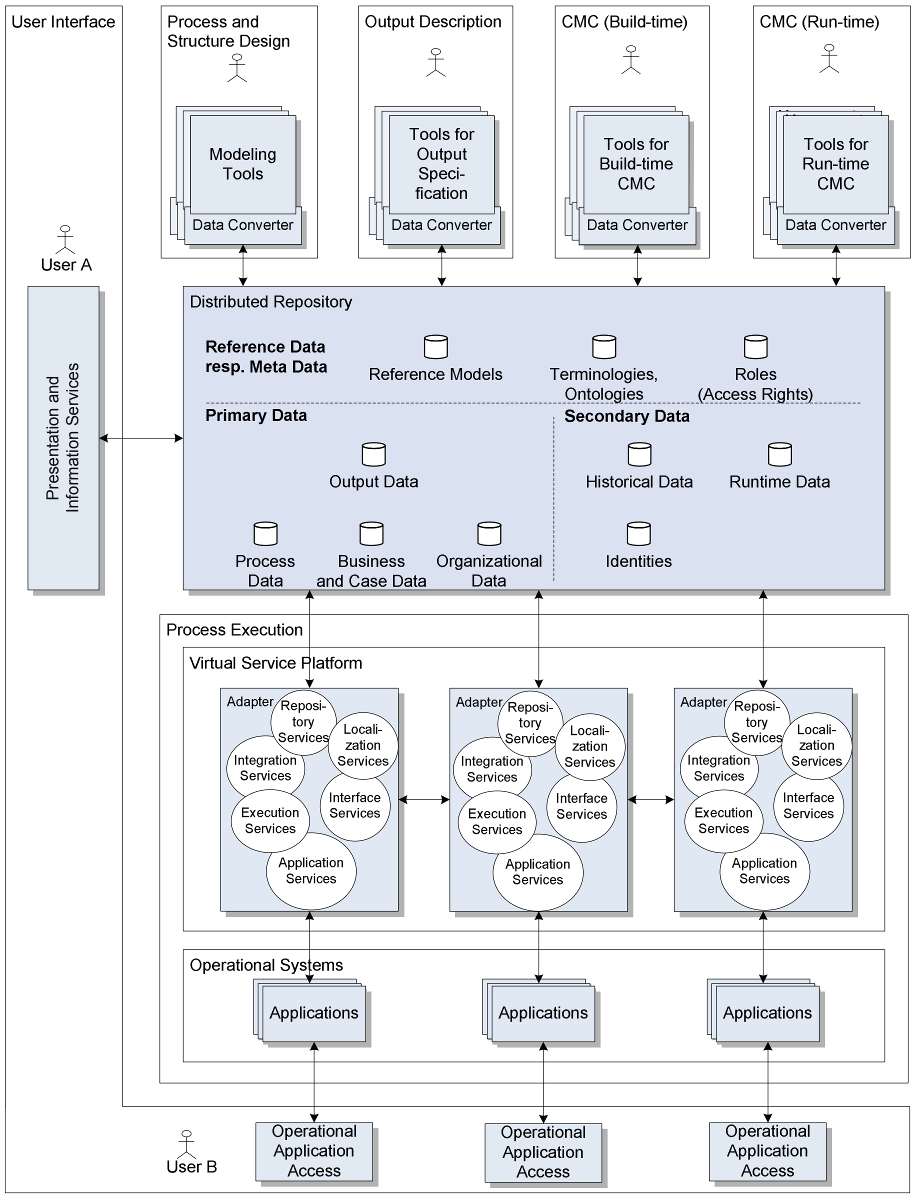


Fig. 1. Architecture for Collaborative Scenarios

organization models the virtual service platform executes processes and integrates different operational systems of the collaboration partners. A life-cycle-model [2] serves as a manual for the process-oriented establishment, the process integration, and operation of collaborations towards a common production of goods and services. The following sections describe the components and characteristics of the architecture.

2.1 Distributed Repository

The architecture is realized upon a physical **distributed repository** managing all data. It enables business process management, common work on the underlying models, and cross-enterprise process execution. Individual knowledge of each partner is stored within decentralized parts of the repository. A logical centralization of the repository represents the knowledge of the overall network.

The repository contains following data categories: ***Reference data resp. Meta data*** provides a basis for the design of process models and organizational models: *Reference models* support the construction of individual models and improve the design of specific organizational structures and processes of the enterprise network [16, 17]. *Ontologies* unify differing vocabulary of concepts and meanings regarding contents and model semantics [18]. Ontologies are conceptual systems of a domain, which obtain knowledge transfer between applications and users. Within the architecture, ontologies integrate different language formats of applications. For the design of models they establish a common conceptual understanding. *Roles* are parts of the security concept of the architecture and define templates, which describe requirements on persons within the network. A role bundles access rights on resources and can be assigned to several persons.

Following the distinction of primary and secondary functions in a value chain [19], also the used data can be distinguished into primary and secondary data. ***Primary data*** support the business process execution and are related with the output of a collaboration: *Output data* describes results of collaborative business processes [4] and is required as input for creating process data and organizational data. *Process data* describe process models of the value-added network. In order to keep business secrets, these models are distinguished into local and global models. Local processes are intra-organizational processes of particular network enterprises, which belong to functions, roles and resources within the enterprises, though they have interfaces to the enterprise-external processes and resources. Global processes form the process structure of the overall network by aggregating the local processes at corresponding interfaces. *Organizational data* describe organization structures of the value-added network and can be distinguished into global and local structure models. Global structure models represent enterprises' relationships in the value-added network, whereas local models represent intra-organizational structures. *Business and case data* are, on the one hand, task-oriented resource data of the value-added network, which pass through the processes and will be processed to stand-alone products [7], e. g. documents or technical drawings. On the other hand, business data describe the task and network itself, e. g. data about partner enterprises or cooperation structure. In a dynamic collaboration these data can continually change.

Secondary data support process management and controlling: *Historical data* and *runtime data* collect defined execution data and possible exceptions due to process execution [20]. Historical data are about former processes and can be used to optimize processes of an actual network. Runtime data provides information about the current processes of the value-added network. Both the historical data and the runtime data support collaboration management and controlling (CMC). Finally, *identities* are basic elements for access control on data of the repository. They are digital features of individuals used for their identification at information systems.

2.2 Decentralized Tools for Accessing the Repository

Output Description is realized by distributed available tools for designing output data. Depending on the output, different enterprises with individual processes are brought together to determine the output. The output description is also used for operational tasks like cost calculation, requests for quotation or accounting.

Process and Structure Design (modeling of the value-added net) is realized by tools for modeling and graphical visualization of processes and organization structures. Global models are stored in an abstract level and are centrally available, while local models are more detailed and only available for corresponding organizations. This can be realized either by decentralized storage of these models or by assigning roles and access rights. The created models are stored in the repository. Reference models can be imported from the repository into the modeling tools. Furthermore, ontologies enable consistent semantic modeling.

Collaboration Management and Controlling (CMC) is distinguished into Build-time-CMC and Run-time-CMC. Referring to a Collaborative Business Process Management Lifecycle [2], Build-time-CMC includes several steps of collaboration establishment, while Run-time-CMC encompasses collaboration execution. Following the distinction into a procedural and an organizational view of collaborations, the methods of CMC are divided into process-oriented and organization-oriented tasks.

Organizational tasks of *Build-time-CMC* are for example portfolio analysis, due diligence or boundary management for collaboration-, enterprise- and department-borders. Concerning collaborative processes, a unique project controlling has to be defined. This contains monetary methods like cost planning, revenue planning, budgeting and calculation as well as planning of collaboration-wide processes. For a pre-evaluation of procedural and organizational behavior of the collaboration we suggest a Petrinet-based simulation . Herewith all process models and organizational models can be validated and verified. The simulation can also estimate lead-times, costs and capacity utilizations, and it can deliver useful data to optimize processes and organizational structures a priori.

An important organizational task of *Run-time-CMC* is the steering of the collaboration partners' behavior. Used methods are e. g. collaboration-intern transfer prices or a repertory-grid-based soft-fact analysis to identify cultural weaknesses of individual enterprises concerning their cultural fit with the collaboration. Process oriented Run-time-CMC-tasks are e. g. integrated progress controls, which include process monitoring, capacity control or performance measurement [4]. Concerning CMC in general, the repository can consist of collaboration-internal data as well as external data. The CMC-results can on the one hand be used to optimize the collaborative processes. On the other hand the results are used as a knowledge-base for the modeling of succeeding collaborations.

2.3 Process Execution

A **Virtual Service Platform** enables process execution and integrates different applications of the collaboration partners. Therefore process data and organizational models of the repository are used. By using local adapters, operational systems interact with each other without implementing a central coordinating instance. Corresponding applications are quested using the location service. The repository service reads relevant process parts and further data like business- or output data. Execution services use these process definitions to execute processes using application services provided by distributed operational systems. Hereby integration services convert different data formats. If necessary, integration services access ontologies. When the process execution in an adapter is ended, the next adapter is triggered to start the next process part. Therefore interface services arrange the interaction between the adapters.

Dynamic process changes can be realized by using a predefined concept for handling different disturbance types. This is stored within the repository and is triggered e. g. by an event-handler within CMC. Based on information stored in runtime data, responsible adapters and persons are determined. Finally, several possible ad-hoc-solutions are enabled, like defining an additional process or reconfigure the current process. The amended process can be continued within the adapter.

As a technical realization of the adapters, web service technologies are suitable [9]. Services are available by using Web Service Description Language (WSDL). Web services can interact with Simple Object Access Protocol (SOAP), based on internet protocols like Hypertext Transfer Protocol (HTTP) [21]. Business Process Execution Language for Web Services (BPEL4WS or BPEL) allows to define and execute web service-based workflows within the execution service [22], while Web Services Choreography Description Language (WS-CDL) defines collaboration between adapters [23].

A further part used for process execution is the **user interface**. A user in this case is a person or an organizational unit which fulfils functions and processes within the collaboration. Two different types of users can be distinguished. On the one hand they use presentation services as well as information services to get access to the repository (e. g. technical drawings, process definitions or visualizations of other data). On the other hand they use “traditional” user interfaces of operational systems to fulfill their tasks.

All data and models are stored in repository-wide unique data formats. By using converters different software can be integrated, although it does not support these standards. In particular data formats of process models and organization models have to fulfill several requirements: on the one hand different modeling tools with different modeling languages can be used. On the other hand inter- and intra-organizational models may use different modeling languages. A uni-directional transformation of models not supporting the common process model format will be described in the following section.

3 Transformation of Business Process Models into BPMN

The presented architecture establishes the general framework towards the management and execution of collaborative scenarios. From a conceptual point of view, business processes have proven to be ideal design items in conjunction with the use of graphical methods and tools [24]. Using heterogeneous process notations within collaborative networks, the need for a central process notation standard emerges to reduce communication and transformation complexity. In the architecture (cf. chapter 2) these aspects are especially relevant to the process and structure design of networks with modeling tools and their connection to the distributed repository. While the architecture abstracts from model heterogeneity, the modeling notations, used in the collaboration, may differ from each other. The following solution presents an adaptable approach for transforming enterprise-specific process models into a collaboration-wide notation standard. The Business Process Modeling Notation (BPMN) specification developed by the Business Process Management Initiative (BPMI) provides an appropriate, standardized notation for representing business processes on a conceptual, near-business level [25].

Following, the Event-driven Process Chain (EPC) is chosen as an exemplary process notation for enterprise-specific process models. Other heterogeneous process notations which are applied in collaboration networks may be transformed in the same way as the EPC models in the example. The suggested procedure model for transforming process models into BPMN can be applied to a unidirectional, horizontal mapping of modeling notations. Underlying models ought to describe things at the same conceptual level to permit a horizontal integration. However, the existence of appropriate XML-representations of the involved notations is essential due to the need of formal process representations. This enables automatic interpretation and transformation by the modeling tools. We aim at a smooth transformation process with minor human interaction.

3.1 Meta Model Mapping

Achieving a collaboration-wide **agreement on the meta-models** of each of the process-modeling methods used by the partners, forms the **first step** towards the model transformation. Meta-models are documented for a majority of notations, but are often altered or enhanced by company-specific definitions. The meta-model for the common repository process format has to be defined - due to its creative nature - manually by modeling experts of all partners. Furthermore a common meta-model for modeling methods (e. g. EPC) should be defined in a unique way to ease later transformation and to disregard enterprise-specific adjustments. Thus, the resulting meta-models help to harmonize the vocabulary of the notation constructs (cf. step two) and are a prerequisite for extracting mapping rules, which is done by defining corresponding process objects (cf. step three).

Second, the **usage of terms** has to be **unified** in order to reach a certain degree of semantic interoperability – by implementing semantic comparability and correctness – and to achieve a high model quality. This is realized by using ontologies within the repository of the presented architecture (cf. chapter 2).

The **third step** comprises the **meta-model mapping** as the conceptual transformation specification, realized by the design of object relations between the meta-model of the process notation for the repository and meta-models of process notations used in modeling tools. Therefore elements of one notation are related manually to corresponding elements of another notation. Future transformation rules can be extracted from these relations.

Table **1** presents an extract of the mapping of EPC to BPMN. Non-ambiguous, bidirectional relations are visualized with double-headed arrows, ambiguous, unidirectional ones use simple arrows. If exception rules are necessary for the mapping because of, e.g. the lack of syntactical and semantical unambiguousness, arrows are put in brackets.

Table 1. Object mapping between EPC and BPMN

EPC	uni- / bidirectional mapping	BPMN
Function	✏✂	Activity
Aggregated Function	✏✂	SubProcess
Basic Function	✏✂	Task
Event	✏ (✂)	Start Event
Event	✏ (✂)	Intermediate Event
...	...	...

3.2 Model Transformation

Process models, which have to be stored within the repository, are exported in a **fourth step** of the procedure model to a standardized exchange format – in this example from EPC to the Event-driven Process Chain Markup Language (EPML) [12]. The following code shows a part of the resulting, formal EPML-representation of the process, which is later transformed via meta-model mapping into BPMN.

Example of a formal EPML-representation

```
<?xml version="1.0" encoding="UTF-8"?>
...
<definition>
...
</definition>
<directory name="Root">
<epc epcId="1" name="business_process">
           <event id="1">
             <name>start_event</name>
             <description>start_event</description>
             <graphics>...</graphics>
           </event>
           <arc id ="2" defRef="1" from="1"
              to="3"/>
           <function id="3">
             <name>function_one</name>
             <description>function_one</description>
             <graphics>...</graphics>
           </function>
          ...
</epc>
</directory>
</epml:epml>
```

To export models via XML, also tool-specific interface languages as – for instance – ARIS Markup Language (AML) may be applied, but the complexity of the transformation process rises within networks with the use of proprietary formats. Thus, standardized instruments should be applied to achieve a simple and effective model transformation.

After the export, the **mapping** between two **XML-methods** is executed in a **fifth step**. Based on the rules predefined in step three, the XML-model (EPML) is transformed into another XML-based process markup-language as – for instance – Petri Net Markup Language (PNML) for Petri Nets or Business Process Modeling Language (BPML) for BPMN. In our example, the BPML is used as the target format [26].

As an example for the underlying analysis of transformation execution, the **task** sequence has to be extracted from the EPML-document by the analysis of relations between events (`<event/>`), arc relations (`<arc/>`) and functions (`<function/>`) and has to be transformed into the corresponding BPML-code. The sequence of EPML functions is transformed to the sequential `<bpml:sequence>`-form with the EPC starting event triggering the BPML sequence. Further corresponding objects, whose mapping was defined in step three, are converted into the new format. The definitions of exceptions and ambiguousness have to be considered within the XML-transformation. The results of the transformation are shown in the following code.

Transformation Result

```
<bpml:process name="business_process">
         <bpml:documentation>
         ...
         </bpml:documentation>
         <bpml:sequence>
           <bpml:event activity="function_one"
             name="start_event"/>
           <bpml:action name="function_one"
             operation="request"/>
           <bpml:action name="function_two"/>
           <bpml:action name="function_three"/>
           ...
         </bpml:sequence>
</bpml:process>
```

With the use of XML-data formats to exchange process model data, an eXtendable Stylesheet Language Transformations (XSLT) script, which transforms XML-documents from one format into another, can be implemented.

For the **sixth** and **final step** towards saving processes in the standardized BPMN format, the resulting XML-model (BPML) is **transferred to the repository**. Process data is now stored in a BPMN-compliant format. The information can be – for instance – exchanged between networking partners in a unified way with reasonable efforts towards communication complexity. The collaboration is executed on a common denominator for the conceptual description of business processes within collaborative scenarios.

4 Conclusions and Further Research

Based on the requirements for IT based support of collaboration networks an architecture for cross-enterprise business processes is given. Moreover, the prerequisites for the unique storage of Business Process Models are described with a generic procedure model for business process model transformations. The presented transformation concept provides an approach to

- solve the problem of heterogeneity in business process modeling by presenting a procedure model to gain syntactic model interoperability.
- consider current research efforts towards XML-based representations of business process models, as – for instance – it is done with EPML and BPML.
- take care of forward-looking standardization approaches, as they were presented by the BPMN – and consider at the same time well-known, established modeling techniques as the EPC to decrease investment risk for enterprises by merging "new" with established models.
- describe business model integration efforts on a conceptual level to get an open reference solution, independent of any fixed connection to certain methods. The approach may be adapted to other modeling notations, such as Petri-Nets or Activity Diagrams as far as a corresponding XML-representation is available and transformation makes sense in a syntactical and semantical way.

The approach does not claim completeness in terms of semantic integration and syntactic mapping covered due to the lack of an adequate formal XML-representation of BPMN and further essential research. It focuses rather on a general procedure model that shows how transformation in a unidirectional way can be conducted in order to facilitate the exchange of process models in heterogeneous environments. The scope of human interaction is minimized because it is only essential to those preleminary steps which can't be formalized like the agreements on the meta-model (cf. first step) or the mapping of meta-models (cf. third step). The model transformation itself may be executed smoothly with appropriate IT solutions (modelling tools, parser etc.). Furthermore, problems as, ambiguity or other textual model defects may be avoided, which leads to a significant reduction in complexity and enables a more efficient planning- and design-task concerning BPM.

Further research is to enhance XML-based representations of standardized modeling notations. Thus, related representation and transformation approaches as, for instance, XML Metadata Interchange [12] have to be analyzed to improve the presented solution. Also the development of supporting tools, which ease the task of exchanging process models between different enterprises, i.e. to automate all possible mapping tasks by adequate rule-based systems is to be enforced. Furthermore, the vertical integration of process information through transformation and mapping of business concepts into IT-interpretable, formal process specifications is another field for further research. This comprises on the one hand a network-wide controlling for tracing processes over several interaction tiers. On the other hand reference models should be designed to facilitate the establishment of cooperation. At the same time a generic connection to infrastructures built on the augmented Service Oriented Architecture paradigm should be created. The presented approach and further research questions are discussed within the background of the research project ArKoS, raised by the German Ministry of Education and Research.

References

[1] Picot, A., Reichwald, R., Wigand, R. T.: Information, Organization and Management: Expanding Markets and Corporate Boundaries. John Wiley & Sons, New York (1999)
[2] Adam, O., Hofer, A., Zang, S.: Cross-enterprise Process Orchestration - Framework for Collaborative Process Automation. In: Proc. 1st International Workshop on Computer Supported Activity Coordination, CSAC 2004, In conjunction with ICEIS 2004 (2004) 185-197
[3] Gronau, N.: Collaborative Engineering Communities - Architecture and Integration Approaches. In: Proc. of the 2003 Information Resources Management Association International Conference (2003) 792-795
[4] Scheer, A.-W.: ARIS, Business Process Frameworks. 3 edn. Springer-Verlag (1999)
[5] van der Aalst, W. M. P.: Making Work Flow: On the Application of Petri Nets to Business Process Management. In: Proc. Applications and Theory of Petri Nets 2002: Proc. of 23rd International Conference, ICATPN 2002 (2002) 1-22
[6] Wild, R. H., Griggs, K. A., Li, E. Y.: A Web Portal/Simulation Architecture to Support Collaborative Policy and Planning Decision Making. In: Proc. Ninth Americas Conference on Information Systems (2003) 2400-2411
[7] Hollingsworth, D. The Workflow Reference Model. Workflow Management Coalition [Online]. Available: http://www.wfmc.org/standards/docs/tc003v11.pdf
[8] Linthicum, D. S.: Enterprise Application Integration. 4 edn. Addison-Wesley, Boston et al. (2003)
[9] Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web Services - Concepts, Architectures and Applications. Springer, Berlin et al. (2004)
[10] Pallos, M. S.: Service-Oriented Architecture: A Primer. EAIJournal 2001 (2001) 32-35
[11] Object Management Group. OMG Model Driven Architecture.[Online]. Available: http://www.omg.org/mda/
[12] Mendling, J., Nüttgens, M.: Exchanging EPC Business Process Models with EPML. In: Proc. 1st GI Workshop "XML4BPM - XML Interchange Formats for Business Process Management" at Modellierung 2004 (2004)
[13] W3C. XSL Transformations (XSLT). W3C [Online]. Available: http://www.w3.org/TR/xslt
[14] Kanungo, S.: Using Systems Dynamics to Operationalize Process Theory in Information Systems Research. In: Proc. Twenty-Fourth International Conference on Information Systems (2003) 450-463
[15] Dedrick, J., Gurbaxani, V., Kraemer, K. L.: Information Technology and Economic Performance: A Critical Review of the Empirical Evidence. ACM Computing Surveys 35 (2003) 1-28
[16] Mertins, K., Bernus, P.: Reference models. In: P. Bernus, K. Mertins, and G. Schmidt, (eds.): Handbook on Architectures of Information Systems. Springer (1998)
[17] Fettke, P., Loos, P.: Classification of reference models - a methodology and its application. Information Systems and e-Business Management 1 (2003) 35-53
[18] Kishore, R., Sharman, R., Ramesh, R.: Computational Ontologies and Information Systems: I. Foundations. CAIS - Communications of the AIS 14 (2004) 158-183
[19] Porter, M. E.: Competitive Advantage - Creating and Sustaining Superior Performance. The Free Press, New York (1985)
[20] Neumann, S., Probst, C., Wernsmann, C.: Kontinuierliches Prozessmanagement. In: J. Becker, M. Kugeler, and M. Rosemann, (eds.): Prozessmanagement - Ein Leitfaden zur prozessorientierten Organisationsgestaltung. Springer-Verlag (2003) 309-335

[21] Blake, M. B.: Coordinating Multiple agents for worklow-oriented process orchestration. ISeB - Information Systems and e-Business Management 1 (2003) 387-404
[22] Shapiro, R.: A Comparison of XPDL, BPML and BPEL4WS. ebPML.org, (2002)
[23] W3C: Web Services Choreography Description Language Version 1.0. W3C,, W3C Working Draft (2004)
[24] Champy, J.: X-Engineering the corporation : reinvent your business in the digital age. Hodder & Stoughton, London (2002)
[25] Owen, M., Raj, J.: BPMN and Business Process Management. (2003)
[26] Arkin, A.: Business Process Modeling Language. BPMI.org, (2002)

A Flexible and Modular Framework for Implementing Infrastructures for Global Computing*

Lorenzo Bettini[1], Rocco De Nicola[1], Daniele Falassi[1], Marc Lacoste[2], and Michele Loreti[1]

[1] Dipartimento di Sistemi e Informatica, Università di Firenze
{bettini, denicola, falassi,loreti}@dsi.unifi.it
[2] Distributed Systems Architecture Department, France Telecom R& D
marc.lacoste@rd.francetelecom.com

Abstract. We present a Java software framework for building infrastructures to support the development of applications for systems where mobility and network awareness are key issues. The framework is particularly useful to develop run-time support for languages oriented towards global computing. It enables platform designers to customize communication protocols and network architectures and guarantees transparency of name management and code mobility in distributed environments. The key features are illustrated by means of a couple of simple case studies.

1 Introduction

Technological advances of both computers and telecommunication networks, and development of more efficient communication protocols are leading to an ever increasing integration of computing systems and to diffusion of "Global Computers" [9]. These are massive networked and dynamically reconfigurable infrastructures interconnecting heterogeneous, autonomous and mobile components, that can operate on the basis of incomplete information.

Global Computers are thus fostering a new style of distributed programming that has to take into account variable guarantees for communication, cooperation, resource usage, security policies and mechanisms, and in particular *code mobility* [10, 24]. They have stimulated the proposal of new theories, computational paradigms, linguistic mechanisms and implementation techniques for the design, realization, deployment and management of global computational environments and applications.

We have thus witnessed the birth of many calculi and kernel languages (see, e.g., [7] and the references therein) intended to support global computing programming and to provide tools for formal reasoning over the modelled systems. Many implementations of these formalisms have been proposed, and very often the language used for implementation is Java because it provides many useful features for building network applications with mobile code. However, these Java mechanisms still require a big programming effort, and so they can be thought of as "low-level" mechanisms. Because

* This work was funded by EU Global Computing initiative, project MIKADO IST-2001-32222.

of this, many existing Java-based distributed systems (see, e.g., [1, 4, 8, 19, 22, 23] and the references therein) tend to re-implement from scratch many components that are typical and recurrent in distributed and mobile applications.

To support the implementation of languages for global computing, we have been working on a generic framework called IMC (*Implementing Mobile Calculi*) that can be used as a kind of middleware for the implementation of different distributed mobile systems. Such a framework aims at providing the necessary tools for implementing new language run-time systems directly derived from calculi for mobility. The basic idea and motivation of this framework is that the implementer of a new language would need concentrating on the parts that are really specific of his system, while relying on the framework for the recurrent standard mechanisms. The development of prototype implementations should then be quicker and the programmers should be relieved from dealing with low-level details. The proposed framework aims at providing all the required functionalities and abstractions for arbitrary components to communicate and move in a distributed setting.

IMC provides concrete implementations for the standard and most used functionalities that should fit most Java mobile framework requirements (e.g., Java byte-code mobility and standard network communication mechanisms). A user of IMC can customize parts of the framework by providing its own implementations for the interfaces used in the package. In this respect, the IMC framework can be straightforwardly used if no specific advanced feature is needed. The framework is however open to customizations if these are required by specific mobility systems. Customization of the framework can be achieved seamlessly by taking advantage of design patterns such as *factory method*, *abstract factory*, *template method* and *strategy* [14] that are used throughout the packages.

The framework was designed to achieve both *transparency* and *adaptability*. For instance, for code mobility, the framework provides all the basic functionalities for making code mobility transparent to the programmer: all issues related to code marshalling and code dispatch are handled automatically by the classes of the framework. Its components are designed to deal with object marshalling, code migration, and dynamic loading of code. The framework can also be adapted to deal with many network topologies (flat, hierarchical, peer-to-peer networks, etc.) and with message dispatching and forwarding. Furthermore, the implementer can build his own communication protocols by specializing the protocol base classes provided by the framework. Thus, the developer will only have to implement the parts that are relevant to the system he wants to build: typically, he will develop the communication protocol which best matches application-specific requirements. Connections and network topology are dealt with directly from within the framework. However, the developer may access the current state of his application with listeners to events that the classes of the framework generate.

The main intent of the IMC framework is not to be "yet another" distributed mobile system. It should rather be seen at a meta-level, as a framework/toolbox for building "yet another" distributed mobile system. A primordial version of the IMC framework was initiated within the MIKADO project [6]. The version we present here is completely re-designed and re-implemented to improve usability and to provide many additional features.

2 Overview of the IMC Framework

We now sketch the main functionalities and interfaces of the framework. For the sake of simplicity, we will not detail all the method signatures, e.g., we will not show the exceptions.

2.1 Communication Protocols

When implementing a distributed system, one of the system-specific issues is the choice of the communication protocol, which may range from high-level protocols such as Java RMI, well integrated with the Java Virtual Machine environment and taking advantage of the architectural independence provided by Java, to protocols closer to hardware resources such as TCP/IP. Marshalling strategies may range from dedicated byte-code structures to Java serialization. A generic communication framework [12, 13, 15, 18, 21] should be minimal, and allow to introduce support for new protocols with little effort, without need to re-implement a new communications library. Thus, IMC provides tools to define customized protocol stacks, which are viewed as a flexible composition of micro-protocols. The IMC design, inspired from the *x*-kernel [17] communication framework, allows to define *bindings* with various semantics, and to combine them in flexible ways. Thus, IMC enables to achieve adaptable forms of communication transparency, which are needed when implementing an infrastructure for global computing.

In IMC, a *network protocol* like, e.g., TCP, UDP, or GIOP is viewed as an aggregation of *protocol states*: a high-level communication protocol can indeed be described as a state automaton. The programmer implements a protocol state by extending the `ProtocolState` abstract class and by providing the implementation for the method `enter`, which returns the identifier of the next state to execute. The `Protocol` class aggregates the protocol states and provides the *template method* [14] `start` that will execute each state at a time, starting from the first protocol state up to the final one. Thus, the programmer must simply provide the implementation of each state, put them in the correct order in a protocol instance, and then start the protocol.

```
public class Protocol {
  public void start() { /* executes the states */ }
}
```

```
public abstract class ProtocolState {
  public abstract String enter();
}
```

The protocol states abstract away specific communication layers. This enables re-using of a protocol implementation independently from the underlying communication means: the same protocol can then be executed on a TCP socket, on UDP packets or even on streams attached to a file (e.g., to simulate a protocol execution). This abstraction is implemented by specialized streams: `Marshaller` (to write) and `UnMarshaller` (to read). These streams provide high-level and encoding-independent representations of messages that are about to be sent or received, i.e., they are basically an extension of standard `DataOutput` and `DataInput` Java streams, with the addition of means to send and receive migrating code (explained later) and serialize and deserialize objects. The interface of `UnMarshaller` is the following (the interface of `Marshaller` contains the corresponding write instead of read methods):

```
public interface UnMarshaller extends DataInput, Closeable, MigratingCodeHandler {
  public Object readReference();
  public MigratingCode readMigratingCode();
  public MigratingPacket readMigratingPacket();
}
```

The data in these streams can be "pre-processed" by some customized *protocol layers* that remove some information from the input and add some information to the output: typically this information are protocol-specific headers removed from the input and added to the output. A protocol layer is an abstract representation of a communication channel which uses a given protocol. It lets messages be sent and received through the communication channel it stands for using that protocol. The base class `ProtocolLayer` deals with these functionalities, and can be specialized by the programmer to provide his own protocol layer.

These layers are then composed into a `ProtocolStack` object that ensures the order of preprocessing passing through all the layers in the stack. For instance, the programmer can add a layer that removes a sequence number from an incoming packet and adds the incremented sequence number into an outgoing packet. The framework also provides functionalities to easily implement *tunnels*, so that it can be possible, e.g., to implement a tunneling layer to tunnel an existing protocol into HTTP (see Section 3).

Before reading something from a stack, a protocol state must obtain an `UnMarshaller` instance from the stack by calling the method `up`: this allows the stack layers to retrieve their own headers. In the same way, before starting to write information to the network, the state must obtain a `Marshaller` instance from the stack by calling the method `prepare`, so that the stack layers can add their own headers into the output. When the state has finished to write, it must notify the stack by calling the method `down`, passing the marshaller instance it had used to write the information, in order to flush the output buffer.

The methods `up`, `prepare` and `down` rely on methods `doUp`, `doPrepare` and `doDown`, respectively, of the class `ProtocolLayer`, that will be called in the right order so to implement the stack of layers. The subclasses of `ProtocolLayer` can provide their own implementations for these methods.

```
public class ProtocolLayer {
  public UnMarshaller doUp(UnMarshaller um) { ... /* implementation of the programmer */ ... }
  public Marshaller doPrepare(Marshaller m) { ... /* implementation of the programmer */ ... }
  public void doDown(Marshaller m) { ... /* implementation of the programmer */ ... }
}
```

The `UnMarshaller` returned by the lower layer in the stack is passed to the implementation method `doUp`; thus, a layer can use the passed `UnMarshaller` to retrieve its own header and pass the `UnMarshaller` to the next layer, or it can create a new `UnMarshaller` to pass to the next layer. The latter scenario is typical of tunneling layers (as briefly shown in Section 3). Similarly, the `Marshaller` returned by the lower layer is passed to `doPrepare`. Typically, the first `UnMarshaller` and `Marshaller` objects will be created by the lowest layer, e.g., in case of a TCP socket, it will be a stream attached to the socket itself, while, in case of UDP packets, it will be a buffered stream attached to the datagram contents. Low layers for TCP and UDP are already provided by the framework.

2.2 Code Mobility

This part of the framework provides the basic functionalities for making code mobility transparent to the programmer. It deals with object marshalling, code migration, and dynamic loading of code. An object will be sent along with the byte-code of its class, and with the byte-code of all the classes of the objects it uses (i.e., all the byte-code it needs for execution). Obviously, only the code of user-defined classes must be sent, as other code (e.g., Java class libraries and the classes of the IMC packages) must be common to every application. This guarantees that classes belonging to Java standard class libraries are not loaded from other sources (especially, the network); this would be very dangerous, since, in general, such classes have many more access privileges with respect to other classes. The framework also allows the programmer to manually exclude other classes (or entire packages) from mobility.

The framework defines the empty interface `MigratingCode` that must be implemented by the classes representing a code that has to be exchanged among distributed sites. This code is intended to be transmitted in a `MigratingPacket`, stored in the shape of a `byte` array. How a `MigratingCode` object is stored in and retrieved from a `MigratingPacket` is taken care of by the following two interfaces:

```
public interface MigratingCodeMarshaller {
  public MigratingPacket marshal(MigratingCode code);
}

public interface MigratingCodeUnMarshaller {
  public MigratingCode unmarshal(MigratingPacket p);
}
```

Starting from these interfaces, the framework provides concrete classes that automatically deal with migration of Java objects together with their byte-code, and transparently deserialize such objects by dynamically loading their transmitted byte-code. In particular, the framework provides the base class `JavaMigratingCode`, implementing the above mentioned interface, `MigratingCode`, that provides all the procedures for collecting the Java classes that the migrating object has to bring to the remote site:

```
public class JavaMigratingCode extends Thread implements MigratingCode {
  public JavaMigratingPacket make_packet() {...}
}
```

The method `make_packet` will be used directly by the other classes of the framework or, possibly, directly by the programmer, to build a packet containing the serialized (marshalled) version of the object that has to migrate together with all its needed byte-code. Thus, this method will actually take care of all the code collection operations. The names of user defined classes can be retrieved by means of class introspection

(*Java Reflection API*). Just before dispatching a process to a remote site, a recursive procedure is called for collecting all classes that are used by the process when declaring: data members, objects returned by or passed to a method/constructor, exceptions thrown by methods, inner classes, the interfaces implemented by its class, the base class of its class. Once these class names are collected, their byte code is gathered and packed along with the object in a JavaMigratingPacket object (a subclass of MigratingPacket storing the byte-code of all the classes used by the migrating object, besides the serialized object itself).

Finally, two classes, implementing the above mentioned interfaces Migrating-CodeMarshaller and MigratingCodeUnMarshaller, will take care of actually marshalling and unmarshalling a JavaMigratingPacket containing a migrating object and its code. In particular, the former will basically rely on the method make_packet of JavaMigratingCode, while the latter will rely on a customized *class loader* provided by the framework (a NodeClassLoader) to load the classes stored in the JavaMigratingPacket and then on Java serialization to actually deserialize the migrating code contained in the packet.

The readMigratingCode method of the UnMarshaller, shown in Section 2.1, will rely on an a MigratingCodeUnMarshaller to retrieve a migrating object and the corresponding method in Marshaller will rely on a MigratingCodeMarshaller to send a migrating object, so that all the code mobility issues will be dealt with internally by the framework. Even in this case, the programmer can provide his own implementations of MigratingCodeUnMarshaller and MigratingCodeMarshaller so that the framework will transparently adapt to the customized code mobility. For further details and examples concerning this part of the framework we refer the reader to [2].

2.3 Node Topology

The framework already provides some implemented protocols to deal with *sessions*. The concept of session is logical, since it can then rely on a physical connection (e.g., TCP sockets) or on a connectionless communication layer (e.g., UDP packets). In the latter case, a keep-alive mechanism can be implemented. A SessionManager instance will keep track of all the connections.

This can be used to implement several network topology structures: a *flat* network where only one server manages connections and all the clients are at the same level; a *hierarchical* network where a client can be in turn a server and where the structure of the network can be a tree or, in general, an acyclic graph of nodes; or, a *peer-to-peer* network.

A participant to a network is an instance of the class Node contained in the framework. A node is also a container of running processes that can be thought of as the computational units. The framework provides all the means for a process to access the resources contained in a node and to migrate to other nodes. Thus, a developer of a distributed and mobile code system has all the means to start to implement its own infrastructure or the run-time system for a mobile code language.

A process is a subclass of the class NodeProcess that implements the JavaMigratingCode base class (this allows to easily migrate a process to a remote site), and can be added to a node for execution with the method `addProcess` of the class `Node`.

A `NodeProcess` has the following interface:

```
public abstract class NodeProcess extends JavaMigratingCode {
  public abstract void execute();
  public final void run() { // framework initialization operations; then call execute() }
}
```

Thus, a node keeps track of all the processes that are currently in execution. A concurrent process is started by calling `start` on the `NodeProcess` thread; the final implementation of `run` will initialize the process structure (not detailed here) and then invoke `execute` that must be provided by the programmer.

A different kind of process, called *node coordinator*, is allowed to execute privileged actions, such as establishing a session, accepting connections from other nodes, closing a session, etc. Standard processes are not given these privileges, and this allows to separate processes that deal with node configurations from standard programs executing on nodes. For these processes a specialized class is provided called `NodeCoordinator`.

The programmer can provide its implementation of the concept of `NodeLocation` to address in a unique way a node in the net (e.g., the standard IP address:port representation). If there is a session with a node, then a location is mapped by the session manager into a protocol stack. Thus a process can retrieve a stack to run its own protocols with a remote node.

The framework also provides means to "manipulate" a protocol: it permits extending a protocol automaton by adding new states and extending the protocol stack by inserting new layers. With respect to the manipulation of the protocol automaton, it is possible to add a new starting state and a new final state, so that the original protocol is embedded in an extended protocol. When a new start and a new end state are added to an existing protocol, the framework will also take care of re-naming the previous start and end

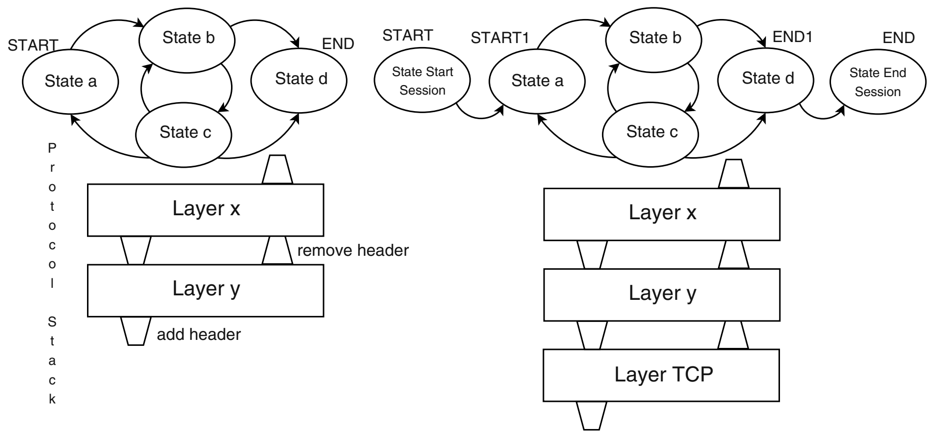


Fig. 1. The original protocol (left) and the new protocol extended with a new start and end state and the TCP layer (right)

state and update all the references to the original start and end state with the re-named version. This will guarantee that the original protocol will transparently work as before internally, while from the outside, the new start state will be executed before the original start state and the new end state will be executed after the original end state.

The manipulation of a protocol is used internally by the classes of the framework, for instance in session management. The `Node` class provides a `connect` method to establish a session and a method `accept` to accept a session. These methods, apart from the connection details (e.g., host and port) also take a protocol instance as a parameter. These methods will take care of establishing (accepting, resp.) a physical connection, add a starting session protocol state as the new start state and a ending session state as the end state to the passed protocol. They also take care of setting the low layer in the protocol stack (e.g., TCP socket or UDP datagrams). Then, the protocol can be started. This manipulation is depicted in Figure 1.

2.4 Naming and Binding

The framework also supports logical name management, inspired by the JONATHAN ORB [12], which provides very flexible primitives to implement the concepts defined in the CORBA naming service [20]. This part of the framework aims to define a uniform manner to designate and interconnect the set of objects involved in the communication paths between computational nodes.

In IMC, an *identifier* is a generic notion of name that uniquely designates an object in a given naming context. Identifier semantics are naming context-specific: distributed, persistent, etc. A *naming context* provides name creation and management facilities. It guarantees that each of the names it controls designates some object unambiguously. It generally maps a name to an object or entity designated by that name, or can also map names to other contexts, if the resolution of names needs to be refined. Finally, a *binder* is a a special kind of naming context that, for a given managed name, is able to create an access path, also called *binding*, towards the object designated by that name.

These definitions offer a generic and uniform view of bindings, and clearly separate object identification from object access: in a given naming context `nc`, a new name for an object `o` is obtained by the `nc.export(o)` invocation. Chains of identifiers can then be created by exporting that name to other naming contexts. Moreover, the creation of an access path to object `o` designated by identifier `id` is performed by the `id.bind()` invocation which returns a ready-to-use surrogate to communicate with `o`. These abstractions are reflected in the following Java interfaces:

```
public interface Identifier {
  public NamingContext getContext();
  public Object bind();
  public Object resolve();
}
```

```
public interface NamingContext {
  public Identifier export(Object obj);
}
```

The `Identifier` interface represents the generic notion of identifier described above. It contains a reference to its naming context, and bears the fundamental `bind` operation to set up a binding between two (possibly remote) objects. The interface, using the `resolve` method, also permits returning the next element in a chain of identifiers, where each identifier was obtained as the result of exporting the next one to some naming context.

An object implementing the `NamingContext` interface stands for the most generic notion of a naming context which manages names of type `Identifier`. The interface includes the `export` operation to create a new name in a given context – which can, if used repeatedly, create chains of identifiers of arbitrary length. Other methods, not represented here, deal with identifier transmission over the network, using encoding-independent representations, namely involving the `Marshaller` and `UnMarshaller` interfaces already described.

This export-bind pattern is closely related to the communication part of IMC: a `Protocol` object can be viewed as a binder which exports (i.e., builds an access path to) a communication end-point, a `ProtocolLayer` designated through a specific type of identifier, namely a protocol layer identifier. Typically, the `export` operation will be called by a server object to advertise its presence on the network. This will be translated into a call to the `accept` method of a `Node` object, to accept incoming network connections. The `bind` operation will be called by a client-side object to bind to the interface designated by a given identifier. This will be translated into a call to the `connect` method of the `Node` object, to establish the communication path to the remote server-side object.

3 Some Examples

In this section we will present some simple examples that show how the framework can be used to program a customized protocol. We will not show all the details of the code, but we concentrate on how the single objects developed by the programmer can be composed together and used from within the framework itself.

First of all, in Listing 1 we show a protocol layer that removes a sequence number from the input stream and writes the incremented sequence number in the output stream. Thus, when a protocol state starts reading, this layer will remove this header and when a state starts writing this layer will add the incremented sequence number. We can create our protocol stack with this layer:

```
ProtocolStack mystack = new ProtocolStack();
mystack.insertLayer(new IncrementProtocolLayer());
```

```
public class IncrementProtocolLayer extends ProtocolLayer {
   private int sequence;
   protected UnMarshaller doUp(UnMarshaller um) {
      sequence = um.readInt(); return um;
   }
   protected Marshaller doPrepare(Marshaller m) {
      m.writeInt(sequence + 1); return m;
   }
}
```

Listing 1: A protocol layer that deals with sequence numbers

```
public class EchoProtocolState extends ProtocolState {
  public String enter() {
    UnMarshaller um = up();   // start reading
    String line = um.readStringLine();
    Marshaller m = prepare();   // stop reading, start writing
    m.writeStringLine(line);
    down(m);   // finish writing
    return "END";
  }
}
```

Listing 2: An echo protocol state

We can now implement our own protocol; for simplicity it will consist of only one state, that does nothing but read a line and send it back (an echo server); after that the protocol ends. In order to implement such a state, we only need to extend the `ProtocolState` base class and provide the implementation for the method `enter` and return the state `END` as the next state in the protocol (Listing 2). We can then create our protocol instance, set the protocol stack, and add the start state:

```
Protocol myprotocol = new Protocol();
myprotocol.setStack(mystack);
myprotocol.setState("START", new EchoProtocolState());
```

The protocol is now built, but no communication layer has been set yet. In order to do so, we can use the `Node` class functionalities:

```
Node mynode = new Node();
mynode.accept(9999, myprotocol);
myprotocol.start();
```

These instructions wait for an incoming connection on port 9999, update the protocol with a starting connection state and a final disconnection state, and update the protocol stack with the low communication layer. At this point, the protocol can start on the established physical connection.

As we suggested in Section 2, the framework provides a specialized protocol layer base class, `TunnelProtocolLayer`, that permits implementing a tunneling layer, for enveloping a protocol inside another one. A typical example is that of an *http tunnel* that wraps a protocol in HTTP requests and responses. Notice that a tunnel layer does not simply remove a header when reading and add a header when writing: typically it will need to read an entire message, strip the tunneling protocol information, and pass the wrapped information to the upper layer; in the same way, it will need to intercept the information written by the upper layer and wrap it into a message according to the tunneling protocol. For this reason the framework provides this specialized base class with the features to implement these more complex functionalities. In particular, `TunnelProtocolLayer` provides two piped stream pairs to allow the tunnel layer to communicate with the tunneled layer: the field `tunneledMarshaller` is piped with the field `newUnMarshaller` (i.e., anything written into `tunneledMarshaller` can be read from `newUnMarshaller`). Thus, the tunnel layer can implement the `doUp` as follows:

```
public class HTTPTunnelLayer extends TunnelProtocolLayer {
  protected UnMarshaller doUp(UnMarshaller um) {
    String data = strip(readHTTPRequest(um));
    tunneledMarshaller.writeStringLine(data);
    return newUnMarshaller;
  }
}
```

Similarly the implementation of `doPrepare` will return to the tunneled layer a piped `UnMarshaller` and `doDown` will read the data written by the tunneled layer from the other end of the pipe, envelop the data in the tunnel protocol structure and pass everything to the lower layer by using the `Marshaller` originally returned by the lower layer's `prepare` method.

Since a tunneling layer is still a layer, it can be inserted smoothly in an existing protocol stack:

```
ProtocolStack mystack = new ProtocolStack();
mystack.insertLayer(new IncrementProtocolLayer());
mystack.insertLayer(new HTTPTunnelLayer());
```

Let us stress that the insertion of the tunnel layer did not require any change to the existing protocol states and layers.

4 Conclusions

We have presented a Java software framework for building infrastructures to support the development of applications over global computers where mobility and network awareness are key issues. The framework enables platform designers to customize communication protocols and network architectures and is particularly useful to develop run-time supports for languages oriented towards global computing.

The components have been designed after a close analysis of proposed models for mobile computing [3, 7]. We have tried to single out the most recurrent notions of network aware programming and packed them together, to permit developers to concentrate on the really specific of their system, while relying on the framework for the recurrent standard mechanisms (node topology, communication and mobility of code).

The main aim of the framework is speeding up the development of prototype implementations and relieving programmers from low level details. Of course, if applications require a specific functionality that is not in the framework (e.g., a customized communication protocol built on top of TCP/IP, or a more sophisticated mobile code management), programmers can customize the part relative to these mechanisms. The presented framework will be shortly released as open source software.

It has been pointed out that our design has some similarities with *.Net Remoting*, which provides an abstract approach to interprocess communication and separates the "remotable" object from a specific client or server application domain and from a specific mechanism of communication. A closer look at the relationships between the two approaches is definitely on demand. These relations are now under investigation.

In the near future, we will also use the framework for implementing $D\pi$ [16] and for re-engineering KLAVA [5], the run time support for KLAIM [11] and for implementing richer languages for global computing. But, we plan also to extend the IMC components to deal with security issues.

Acknowledgments. We are grateful to all people involved in the MIKADO project, in particular, we would like to thank L. Lopes and V. Vasconcelos that contributed to the initial design of IMC.

References

1. A. Acharya, M. Ranganathan, and J. Saltz. Sumatra: A Language for Resource-aware Mobile Programs. In Vitek and Tschudin [25], pages 111–130.
2. L. Bettini. A Java Package for Transparent Code Mobility. In *Proc. FIDJI*, LNCS 3409, pages 112–122. Springer, 2004.
3. L. Bettini, M. Boreale, R. De Nicola, M. Lacoste, and V. Vasconcelos. Analysis of Distribution Structures: State of the Art. MIKADO Project Deliverable D3.1.1, 2002.
4. L. Bettini, R. De Nicola, G. Ferrari, and R. Pugliese. Interactive Mobile Agents in X-KLAIM. In *Proc. WETICE*, pages 110–115. IEEE, 1998.
5. L. Bettini, R. De Nicola, and R. Pugliese. KLAVA: a Java Package for Distributed and Mobile Applications. *Software - Practice and Experience*, 32(14):1365–1394, 2002.
6. L. Bettini, D. Falassi, R. D. Nicola, M. Lacoste, L. Lopes, M. Loreti, L. Oliveira, H. Paulino, and V. Vasconcelos. Language Experiments v1: Simple Calculi as Programming Language. MIKADO Project Deliverable D3.2.1, 2004.
7. G. Boudol, I. Castellani, F. Germain, and M. Lacoste. Models of Distribution and Mobility: State of the Art. MIKADO Project Deliverable D1.1.1, 2002.
8. G. Cabri, L. Leonardi, and F. Zambonelli. Reactive Tuple Spaces for Mobile Agent Coordination. In K. Rothermel and F. Hohl, editors, *Proc. 2nd Int. Workshop on Mobile Agents*, LNCS 1477, pages 237–248. Springer, 1998.
9. L. Cardelli. Abstractions for Mobile Computation. In J. Vitek and C. Jensen, editors, *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, LNCS 1603, pages 51–94. Springer-Verlag, 1999.
10. G. Cugola, C. Ghezzi, G. Picco, and G. Vigna. Analyzing Mobile Code Languages. In Vitek and Tschudin [25].
11. R. De Nicola, G. Ferrari, and R. Pugliese. KLAIM: a Kernel Language for Agents Interaction and Mobility. *IEEE Transactions on Software Engineering*, 24(5):315–330, 1998.
12. B. Dumant, F. Horn, F. Dang Tran, and J.-B. Stefani. Jonathan: an Open Distributed Processing Environment in Java. In *Proc. MIDDLEWARE*, 1998.
13. ExoLab Group. The OpenORB project, 2002. `http://openorb.exolab.org/`.
14. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
15. R. Hayton, A. Herbert, and D. Donaldson. Flexinet: a Flexible Component Oriented Middleware System. In *Proc. ACM SIGOPS European Workshop*, 1998.
16. M. Hennessy and J. Riely. Resource Access Control in Systems of Mobile Agents. In U. Nestmann and B. C. Pierce, editors, *HLCL '98*, volume 16.3, pages 3–17. Elsevier, 1998.
17. N. Huntchinson and L. Peterson. The *x*-kernel: an Architecture for Implementing Network Protocols. *IEEE Transactions on Software Engineering*, 17(1):64–76, 1991.
18. R. Klefstad, D. Schmidt, and C. O'Ryan. The Design of a Real-time CORBA ORB using Real-time Java. In *Proc. ISORC*, 2002.
19. D. Lange and M. Oshima. *Programming and Deploying Java Mobile Agents with Aglets*. Addison-Wesley, 1998.

20. Object Management Group. *Naming Service Specification*, version 1.3 edition, 2004. Available at `http://www.omg.org`.
21. C. O'Ryan, F. Kuhns, D. Schmidt, O. Othman, and J. Parsons. The Design and Performance of a Pluggable Protocols Framework for Real-time Distributed Object Computing Middleware. In *Proc. MIDDLEWARE*, 2000.
22. H. Peine and T. Stolpmann. The Architecture of the Ara Platform for Mobile Agents. In *Proc. MA*, LNCS 1219, pages 50–61. Springer, 1997.
23. G. Picco, A. Murphy, and G.-C. Roman. LIME: Linda Meets Mobility. In D. Garlan, editor, *Proc. ICSE*, pages 368–377. ACM Press, 1999.
24. T. Thorn. Programming Languages for Mobile Code. *ACM Computing Surveys*, 29(3):213–239, 1997. Also Technical Report 1083, University of Rennes IRISA.
25. J. Vitek and C. Tschudin, editors. *Mobile Object Systems - Towards the Programmable Internet*, LNCS 1222. Springer, 1997.

An Architecture for Implementing Application Interoperation with Heterogeneous Systems

George Hatzisymeon[1], Nikos Houssos[2], Dimitris Andreadis[3], and Vasilis Samoladas[1]

[1] Tech. U. of Crete
{george, vsam}@softnet.tuc.gr
[2] Communication Networks Laboratory, University of Athens
nhoussos@di.uoa.gr
[3] JBoss Europe
dimitris@jboss.org

Abstract. We are concerned with the issues faced by software developers with a certain family of distributed applications; those that connect to and interoperate with a heterogeneous infrastructure, i.e., a large heterogeneous collection of external systems (databases, embedded devices, network equipment, internet servers etc.) using different communication protocols. This *product family* includes applications such as e-commerce systems, network management applications and Grid-based collaborations. For such applications, implementing the interoperation logic is both challenging and expensive. We discuss the major concerns that contribute to the problem, such as transaction support, security and management, as well as integration with workflow or component frameworks. We propose an architecture and related development methodology, based on generative programming, to reduce implementation complexity, allow for rapid application development, ease deployment and manageability.

1 Introduction

In recent years, there is an increasing tendency for automation of complicated distributed processes whose realisation has previously included a significant degree of human intervention. Of particular interest to us are those activities that involve multiple administrative domains, depend on heterogeneous infrastructures and are subject to frequent change. Relevant examples can be found in diverse fields like Business-to-Consumer (B2C) and Business-to-Business (B2B) e-commerce, service management and provisioning in wired and wireless networks, Grid-based collaborations and computer-aided manufacturing, to name but a few. One of the most challenging aspects of automation is the seamless cooperation of the application logic with a variety of external systems, such as enterprise applications (e.g., ERP, CRM, Billing), databases, Internet/Intranet servers (e.g., web, email, FTP), and embedded devices (network equipment, sensors, instruments etc.) The development of modules that interoperate with such systems is tedious and time-consuming, since a lot of effort needs to be put on implementing the required communication/access protocols and data transformations.

The aforementioned developer responsibilities are facilitated by tools that enable the programmer to work at a relatively higher level of abstraction, ranging from simple libraries (e.g., email, FTP or SNMP clients) to powerful middleware (e.g., CORBA, JDBC). However, even with the utilisation of such tools, the task remains challenging, for at least three reasons. First, programmers still need to be aware of a variety of different APIs and technologies, which are irrelevant to the actual task to be implemented. Second, the integration of the external systems into the application frequently requires support for advanced features; dynamic pluggability, transactional execution (when it is possible to undo actions), concurrency control, manageability and configurability. Third, to make external systems available to application logic, they must be made accessible via specialized interfaces: as workflow activities (nodes), web services, component objects, and so forth. This is achieved via suitable wrappers that can be tedious to compose and maintain by hand.

The present contribution aims to provide a framework for realising interaction with heterogeneous infrastructures that minimises the effort required for the development of the interaction logic. In particular, it defines a component architecture and related mechanisms that provide the following capabilities:

- Rapid development of "one-of-a-kind" components to interoperate with external systems, based on *generative programming* techniques [10] utilizing an active library [25] of access mechanisms (e.g., telnet, FTP, HTTP, JDBC, SNMP).
- On-the-fly deployment and integration of components with the underlying transaction, management and security infrastructures of the application.
- Access to interoperation components from different types of business logic implementations (e.g., workflows, web/grid services) through generatively created wrappers.

To develop our framework, we were guided by the identification of a *product family*, or *domain* (in the sense of [10, 11]), namely that of applications which interoperate with a large, or frequently changing, collection of heterogeneous systems. In this domain, development and maintenance costs of interoperation are comparable, or even dominate, development and maintenance costs of application logic. Our contribution includes a refinement of the semantic content of access mechanisms/protocols (*domain analysis*, in software reuse parlance) and a proposed domain architecture. To validate our approach, an operational prototype has been developed, making use of commercial component frameworks (JBoss [3, 4]) and software engineering tools (Eclipse platform [6], Velocity generator [5]). The prototype has been successfully employed to provide application interoperation with relational databases, network elements and Internet servers.

The rest of the current document is organized as follows: Section 2 presents related work. Section 3 provides an overview of the proposed architecture and elaborates on vital mechanisms such as the task of integrating into an application atomic functional elements and techniques for their template-based, rapid development through predefined adaptors. Section 4 discusses the main choices and trade-offs involved in the design of our solution. The last section of the paper is devoted to summary/conclusions and identification of important elements for further work.

2 Related Work

Our work addresses interoperability issues of distributed applications composed by a possibly dynamic, heterogeneous collection of external systems. Such issues have been addressed before in fundamental work in distributed systems, especially in the area of middleware. The bulk of the work can be cast into two broad approaches: (a) general-purpose, low-level mechanisms, such as basic middleware, and (b) application-specific, high-level techniques.

The first approach, which is typified by traditional middleware (RPC, CORBA, RMI, etc.) has been broadly studied. The general direction of the work is to abstract IPC and networking facilities into a high-level application framework. Recent progress in this area has broadened applicability in challenging cases, such as real-time and embedded applications [13, 16]. Composition of communication protocols has also been studied, notably in the BAST system [15] and in [23, 26]. These techniques are very broadly applicable, but focus on the communication task, and have not been integrated with the higher-level aspects of application frameworks, such as transaction, security and management. The recent introduction of Web Services has advocated a new style of loose integration of autonomous systems, the so-called Service-Oriented Architecture. The platform is currently being augmented with additional conventions related to high-level application aspects (e.g. transactions [8] and resources [12]). It has also been adopted as the standard paradigm for the development of the Grid [13].

The second approach in system interoperability took an application-oriented view of the problem, where the goal was to integrate external systems as close to the application logic as possible. The most notable advances have been in the area of information system integration. The introduction of widely used wrapper technologies (ODBC, JDBC etc.) allowed uniform access to multiple external systems using high-level languages (such as SQL). This has enabled technologies such as mediator-based information system integration [24] over heterogeneous environments, and object-relational mapping technologies (e.g. Enterprise JavaBeans).

What is needed today is the convergence of the two approaches outlined above: general-purpose, high-level system interoperation mechanisms. Ambitious software engineering efforts (notably OMG's Model Driven Architecture [20]) are underway to combine current techniques. At the same time, an array of component-based application frameworks are being developed for web (JSP), client-server applications (J2EE), web and grid services (e.g. Globus [13]), mobile agents (e.g. Cougaar [16]), peer-to-peer systems (e.g. PROST [21]), bringing forward new generations of large-scale distributed systems. In each of these frameworks there is need for high-level interoperability with external systems, integrated with fundamental transactional, security and management mechanisms. Existing technologies to these directions do exist (e.g. the J2EE Connection Architecture [2]), but they are still little more than hooks into the platform functionality.

3 Architecture

In this section we present our framework in considerable detail. First, we focus on the overall system architecture, introducing fundamental concepts and design. Next, emphasis shifts to application lifetime cycle and the issues thereof.

3.1 Overall Architecture

Access to external systems is accomplished through *actions*, a semantically high-level interface, whose purpose is to isolate application logic from communication and other access concerns as much as possible. Actions have a signature; they accept and return typed arguments, and raise exceptions. Actions must coordinate via concurrency control and transactions. They must implement access control, and perhaps obey other security-related constraints. They must be manageable and discoverable. Finally, they must be accessible in a variety of ways: through workflows, embedded script languages, components (e.g., servlets, EJBs), as exported services, and so forth.

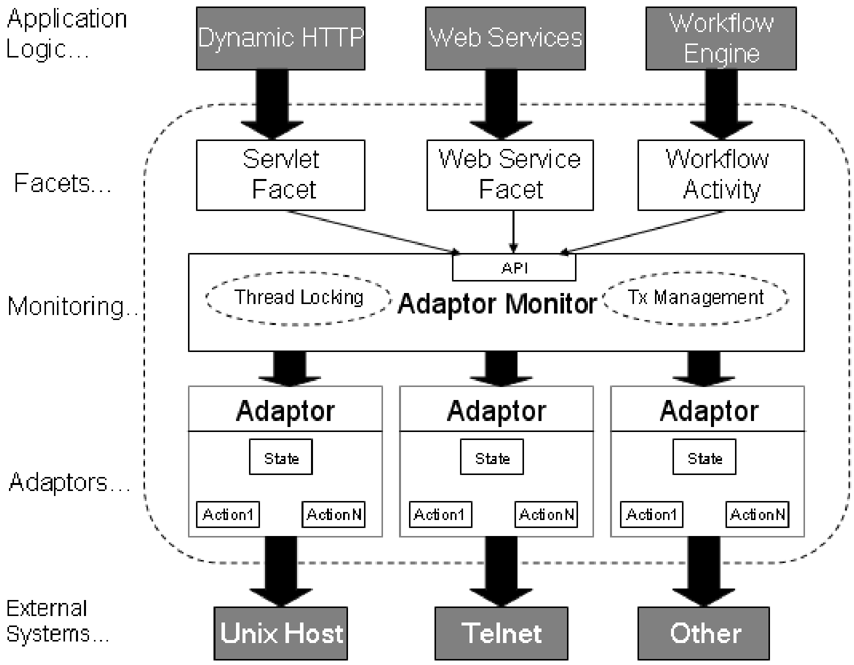


Fig. 1. Making external systems (grey boxes on the bottom) available to application logic (grey boxes on the top): The overall architecture

Based on these concerns we suggest a 3-tier architecture to address the problem. The bottom tier encapsulates external system access specifics: communication channels, protocol implementations, session authorization/login, fault tolerance etc.

The middle tier's purpose is to integrate with the application framework in use (e.g., J2EE, .NET) and provide synchronized and transactional access to the bottom tier. Finally, the top tier implements different interfaces to the lower tiers (workflows, embedded scripting, web service/CORBA/servlet calls, session EJBs etc.). Our proposed architecture along these lines is depicted in Fig. 1.

Adaptors. The logic for application connectivity to external systems is embedded within *adaptors*. An adaptor is a component, which encapsulates the necessary connection state and logic to one or more external systems. Adaptors possess two distinct interfaces. The first is a transactional, high-level interface, consisting of *actions.* This interface is accessed by application logic through the facets (the top layer), and integrates with the underlying application framework, i.e., transaction processing, concurrency control and authorization. The second is a non-transactional, low-level interface, which is only available to the Adaptor Monitor (the middle layer). This interface is used to perform management operations on the adaptor (e.g., resource and connection management and monitoring, security, auditing, on-line configuration and lifecycle management).

Adaptors can relate to external systems or services in a variety of ways. For example, an adaptor may encapsulate a telnet session to a remote Unix host, a TCP/IP multicast group, a Kerberos-authenticated database session, an SNMP-managed device, etc. As a general principle, adaptors are protocol-oriented; they derive from protocol templates, specialized and refined appropriately to comply with application requirements.

Actions. Actions correspond to operations on external systems. Each action is contained within a specific adaptor. Actions are stateless components whose invocations are atomic with respect to application transactions; thus it is desirable that they map to atomic operations on the external system. Each action is specified by a pair of procedures, the first procedure implementing the operation, and the second, which is optional, reversing the operation. These two procedures correspond to the well-known DO-UNDO transactional protocol [16].

In contrast to adaptors, which relate closely to the external system, actions relate to the application logic. Consider for example an adaptor encapsulating a telnet session to a Unix host. The adaptor is responsible for communication-level properties, such as IP address and port, session authentication (login/password exchange), configuration of the conversational exchange (e.g., recognising the session's command prompt), etc. Actions related to this particular adaptor are totally application-specific. For example, if the purpose of connecting to this Unix host is to perform user management on it, sample actions for this adaptor would include `adduser, deluser, chgpass,` and `chgshell.` The developer would be responsible for implementing these actions (and their reversals) as required by the host, e.g., compose the command line necessary to add/delete a user, and parse the command output. The adaptor will only provide a protocol-specific API (e.g., in our example, an `execute` function, accepting a command line and returning a stream of the command output).

Adaptor Monitor. Actions are invoked only via the *Adaptor Monitor*. This module constitutes the middle layer of our architecture and is responsible for application-wide adaptor integration. Primarily, it is a Transaction Processing monitor [16] for action invocations (hence its name). It logs the information needed to reverse the sequence of actions of an aborted transaction. This mechanism integrates closely to the application framework.

Concurrency control for action invocations is supported in cooperation with the TP monitor, by two locking mechanisms: (a) *synchronization locking*, ensuring mutual exclusion among concurrent action invocations from multiple threads, and (b) *Consistency locking*, where transactions can explicitly obtain long-term locks on specific actions, that will preclude other transactions from invoking them until the locks are released. This mechanism can be used to implement transaction scheduling policies, such as serialization [16].

The Adaptor Monitor offers directory services over the deployed adaptors and actions. Apart from name-based discovery it also provides metadata services for adaptors and actions, both in human-readable form (e.g., to be used by interactive management tools), and API-based (i.e., reflection descriptors of adaptor and action interfaces). Another responsibility of the Adaptor Monitor is adaptor lifecycle management: The adaptor interface includes four mandatory operations: `init`, `start`, `stop` and `destroy`. Typically, these are automatically invoked upon particular management operations (e.g., adaptor (re-)deployment, system exit/start). During lifecycle operations, the Adaptor Monitor takes into account global sequencing constraints for setting up adaptors. The relevant information is provided at adaptor design time.

Facets. Facets are responsible for application-wide action integration. The Adaptor Monitor has a standard interface for all adaptor and action-related operations, which may not be convenient to call directly from application logic. Some useful types of facets include:

- Workflow facets. Make actions available as *activities* (workflow nodes) to a workflow engine executing in the application.
- Services facets. Actions/sets of actions become available as Web Services, CORBA or RMI objects etc. to the application and its clients.
- Script facets. Actions become available to application-embedded script languages (e.g., Visual Basic, Python).
- Unit testing facets. Interfacing to the testing and debugging tools.
- Servlets, Java Beans, and other types of application logic components.

Facets are generated automatically from adaptor specifications, using specialized tools for each facet type.

3.2 Implementing Adaptors

Adaptors can be very complex components. Their implementation is in most cases based on the knowledge of a specific protocol/domain/language. To avoid development of every new adaptor from scratch, we employ a generative approach that allows for rapid, simplified implementation. Our approach is based on the development of an active library [25] of protocols, i.e., a collection of protocol implementation templates, which encapsulate most of the required connection knowledge, and can be customized and refined through a graphical tool.

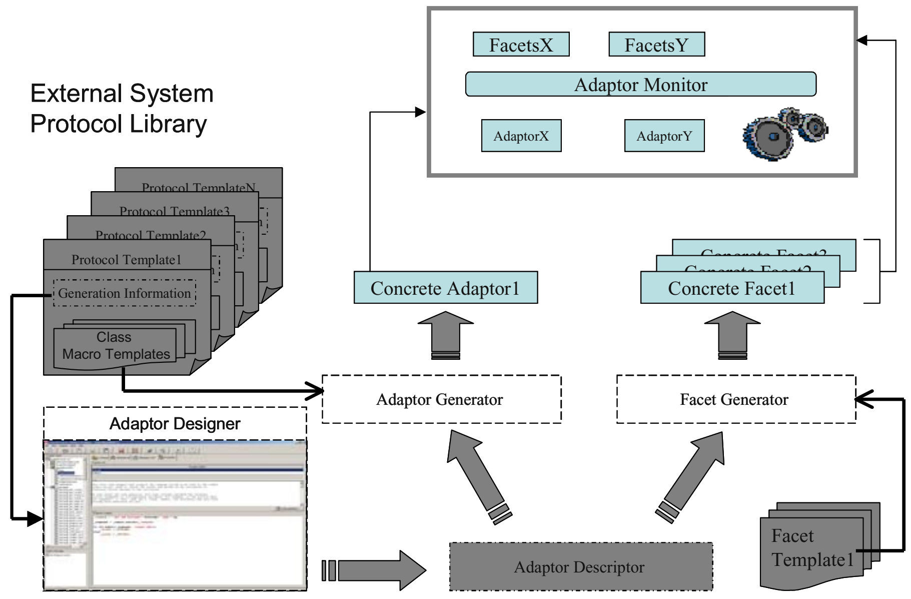


Fig. 2. Adaptor and facet development process

Our adaptor development process is depicted in Fig. 2. The first stage is adaptor design, and is performed graphically using the Adaptor Designer. It comprises of three steps: (a) selection of a protocol template, (b) customization of connection, deployment, lifecycle, authentication and auditing aspects, and (c) implementation of the actions required by the application, which includes, for each action, definition of its signature, implementation of the DO-UNDO logic, locking specification and documentation. The result of this process is an (XML-encoded) Adaptor Descriptor, which is used to drive code generation.

Each protocol template comprises a number of class macro-templates, which contain templatised source code to be fed into the generator, as well as the Generator Information (GI), an XML descriptor of the protocol template. The GI is used to customize the Adaptor Designer to the specific needs of the protocol at hand. It contains a variety of information:

- *Protocol information*, such as name, API exposed by the protocol implementation and deployment information (e.g., dependencies on external software libraries)
- *Adaptor properties*: typed attributes exposed to the adaptor API. These can be the mandatory attributes that maintain the state of the adaptor or additional information required to configure protocol-related operation (connection, resource and lifecycle management, authentication etc.)
- *Action types*: To simplify implementation of actions, each adaptor can support a number of action types. Each action type is specified by a name, a collection of class macro-templates and human-readable documentation of the contract to be supported by action implementations. The contract of an action type consists of

mandatory in-out arguments and guidelines that act as a reference for coding an action's DO-UNDO procedures, and per action action-type. Action types can provide utilities assisting the most common types of interaction processing (text/XML/URL parsers, data transformers, macro expanders, etc.)

The output of Adaptor Designer is an **Adaptor Descriptor** (AD), an XML document holding the specification of a concrete adaptor. It provides code generators with a variety of information that they utilise to parameterise the instantiation of code macro-templates. An AD contains data for both the protocol (mostly copied from the GI) and the adaptor. The main part of adaptor information is a list of *adaptor properties*. These correspond to the attributes defined in the GI, with concrete values provided by the developer. Optionally, additional properties can be specified during adaptor design. Properties can be used on action method implementations, or can be part of the non-transactional adaptor interface.

Important elements of the AD are the *action specifications*. Each action specification contains action type, name, action signature, a human-readable documentation of the action interface and semantics, implementations for the DO-UNDO procedures, specification of locking behavior, and deployment information (e.g., dependency on external libraries). An AD also encapsulates *additional XML metadata* that is associated with the adaptor and individual actions, whose semantics are opaque to the framework. This metadata can be accessed both during facet generation, and at runtime through the Adaptor Monitor.

The last step in adaptor implementation is automated by a code generation tool, which receives the Adaptor Descriptor, and uses the code macro-templates of the protocol template to produce source code, deployment metadata, scripts etc. Facets are also generatively produced by the **facet generator**, based on the adaptor descriptor and a list of appropriate class macro-templates, drawn from the active library.

3.3 Prototype Implementation

For an initial implementation of our platform we chose the Java 2 Enterprise Edition platform (J2EE) and the JBoss application server. JBoss provides robust pluggable implementations of Java Management eXtensions (JMX) [1] and the Java Connector Architecture (JCA) [2]. Adaptors are implemented as standard JMX MBeans, providing an interface accessible through JMX.

We have used the Velocity generator [5] to implement both the Adaptor Generator and the Facet Generator. Velocity provides an intuitive macro language that adds only marginal complexity to the coding of protocol templates. The Adaptor Designer is currently a stand-alone Java application, although we plan to implement a new version inside the Eclipse IDE.

We have implemented a moderate library of protocols, including most Internet services (telnet, FTP, http, SMTP, SNMP), as well as three facet types: an Enterprise Java Bean (EJB) facet, where actions are available to EJBs as methods, a Web Service facet, where actions are exposed as Web Services by JBoss, and a jBPM workflow facet, where actions are available as workflow activities.

4 Discussion

The present section provides a discussion on the architecture proposed in this paper. First, we consider its applicability in two different domains: telecommunications service provisioning and grid-based applications. Then, we elaborate on important choices and trade-offs that we faced in the course of the system design.

4.1 Application Areas

Service Provisioning. The proposed architecture is particularly suited for service provisioning applications, in the general area of telecommunications Operation Support Systems (OSS). The goal of service provisioning is to automate the provisioning of telecommunication services across different technology domains (traditional land line phone service, internet access, mobile access, etc.) [9]. Some of the major challenges of service provisioning addressed the proposed architecture are as follows:

- *Heterogeneity*: Providers offer services over a variety of telecommunication equipment and technologies. A typical provisioning scenario may involve interaction with a dozen different devices or management systems.
- *Consistency*: Activation failures are common in complex systems and they can easily result to wasting valuable network resources if a multi-step activation scenario fails at some intermediate point. Transactional interactions with network devices eliminate the error-prone practice of coding rollback logic by hand.
- *Constant change*. Every so often, the marketing department will come up with yet another bundle of services sold as a package, at which point the activation flow will need to be adapted or extended. Our architecture matches those requirements because it allows rapid, easy introduction of new actions, or alteration of old ones.

Grid Computing. Grids [13] constitute virtual computation platforms, promising to make available unparalleled levels of computing, storage and communication resources to scientific, engineering and business applications. To fulfil this promise, Grid technology must be able to harness the resources contributed by the participants of a virtual organization. These resources form a heterogeneous infrastructure, the *Grid fabric*, which must be made accessible to Grid development and application frameworks through a uniform interface, the *Grid middleware*. Grid-related research has been concerned with the grid middleware and higher-level components: resource management, brokering, semantic discovery, etc. There is relatively little work on integrating fabric resources to grid middleware. In real Grids this is done in ad-hoc ways, with significant cost. Our proposed architecture can reduce this cost, by exposing the Grid fabric to the Grid middleware through adaptors. Thus we can benefit in several ways. Access to resources, applications and datasets, can automatically integrate with transactional, concurrency, semantic/metadata and security mechanisms of the Grid middleware. Semantic issues are of particular interest; brokering and planning performed by Grid middleware depends on a semantic representation of the grid fabric stored in metadata repositories. Suitable facets can be used to easily populate these repositories with minimum effort.

4.2 Design Choices and Trade-Offs

A principal goal of our solution is achieving separation of concerns with regard to development of interoperation logic. This is accomplished through the complementary contribution of three types of actors:

- *Framework developer*: implements functionality common to all adaptors as well as the development tools (e.g., generators, facet templates), as outlined in this paper. Once the framework is available there is little need for subsequent modifications.
- *Connectivity experts*: develop specific protocol templates. The implementation of these templates is tedious and requires extensive knowledge of communication and access protocols (e.g., SMTP, FTP, TELNET, JDBC). It is expected that new protocol templates are continuously needed, albeit with moderate frequency.
- *Application domain experts*: responsible for the application-specific intelligence, i.e., instantiation of adaptors and implementation of actions. This task is normally the easiest and less costly in terms of effort and time. Actions are constantly updated/added to the system, possibly at a high frequency.

The above distinction of roles enables new pieces of connectivity logic to be easily added to an application, so that interoperation requirements are rapidly satisfied.

An important design choice is the dual interface exported by the adaptors, as elaborated in section 3.1. Actions comprise the high-level portion of the adaptor interface, supporting features like transactions, concurrency and authorisation. The rest of the interface is too low-level for the application logic to be aware of; it is available only to the Adaptor Monitor and pertains to management functions. Features like transactionality and concurrency are not supported for these operations; this would considerably complicate matters without any significant benefit. There is ample precedent justifying our choice, e.g., in database systems, where the Data Manipulation Language is transactional, while the Data Definition Language is not.

The ultimate objective of the framework is to enable application logic to invoke actions. An action encompasses only the logic that needs to be executed at the external resource; it does not care how the connectivity is obtained. Furthermore, actions are atomic; they do not encapsulate any further nested actions that can be handled as distinct functions from a transactional point of view. Thus, they need not maintain any state information. Support for transactional behavior is optional. Actions are therefore extremely lightweight components; the simpler among them may consist of only a few lines of code. This leads to minimal effort and time required from the part of the action developer, as well as minimal overhead for the execution of actions. In case of non-reversible actions, the overhead is even smaller, since no invocation history need be preserved.

In designing the transaction support for the Adaptor Monitor, we chose to select DO-UNDO semantics, instead of the more powerful DO-UNDO-REDO semantics. Thus, it will be difficult to implement advanced buffering/caching/coalescing behaviour at the action level. This choice limits performance in a few cases; for example, an object-relational mapping of an external database may be less efficient. On the other hand, we gain in simplicity: for most external systems, the meaning of

REDO is not obvious. A related concern concerns our choice of locking semantics. We chose not to constrain the user to a specific protocol (an obvious choice would have been two-phase locking) but instead allow application logic to control locking explicitly. If more constrained behaviour is desirable for some adaptors, it can in principle be supported by special facets.

A concern we faced during the design of the overall architecture is the management of events that originate from the underlying infrastructure and are of interest to the application. Relevant issues have been the subjects of extensive research efforts in areas related to distributed systems [27, 28]. The approach adopted by our framework so far does not include an explicit mechanism for event propagation towards the application. However, this can be achieved through polling at the application logic level.

5 Summary – Future Work

In this paper, we presented an architecture for application interoperation with heterogeneous infrastructures. Our contributions pertain to applications which have extensive and frequently changing requirements for connection to external resources. Our architecture promotes separation of concerns in the development of interconnection functionality, with a bias in the direction of reducing the burden on the developer of application logic.

With regard to future work, our top priorities include: (a) incorporating event management into the framework, (b) utilisation of the framework in Grid applications based on the Globus platform, and (c) investigation of the architecture implementation on platforms other than J2EE, such as the Cougaar agent framework.

References

1. Java Management Extensions White Paper: Dynamic Management for the Service Age. http://java.sun.com/products/JavaManagement, 1999.
2. J2EE Connector Architecture Specification, Version 1.5, Nov. 2003.
3. JBoss Open Source Application Server, http://www.jboss.org.
4. M Fleury, F Reverbel, The JBoss Extensible Server, International Middleware Conference (Middleware 2003), Brazil, June 2003.
5. Velocity Template Engine, http://jakarta.apache.org/velocity.
6. Eclipse Integrated Development Environment, http:// www.eclipse.org.
7. A. Beugnard, Communication Services as Components for Telecommunication Applications, In Proc. Objects and Patterns in Telecom Workshop (in ECOOP'00), 2000.
8. L. F. Cabrera, G. Copeland, M. Fwingold et al. Web Services Atomic Transaction (WS-AtomicTransaction), Nov 2004.
9. A. Clemm, F. Shen and V. Lee, Generic Provisioning of Heterogeneous Services—a Close Encounter with Service Profiles. Computer Networks 43 (2003), 43-57.
10. K. Czarnecki and U. W. Eisenecker, Components and Generative Programming. In Proc. 7th European Software Eng. Conf., 1998.

11. A. Egyed, N. Mehta and N. Medvidovic, Software Connectors and Refinement in Family Architectures. In Proc. 3rd Int'l W. on Development and Evolution of Software Architectures for Product Families, LNCS 1951, 96-105, 2000.
12. I. Foster, J. Frey, S. Graham et al. Modelling Stateful Resources with Web Services. Preliminary whitepaper version 1.1, 3/5/2004.
13. I. Foster, C. Kesselman, J. Nick, S. Tuecke. Grid Services for Distributed System Integration. Computer, 35(6), 2002.
14. A. Gokhale and D. C. Schmidt, Techniques for Optimizing CORBA Middleware for Distributed Embedded Systems. In Proc. of INFOCOM '99, 1999.
15. B. Garbinato and R. Guerraoui, Flexible Protocol Composition in Bast, In Proc. Int'l Conf. on Distributed Computing Systems (ICDCS), 1998.
16. J. Gray and A. Reuter, Transaction Processing: Concepts and Techniques. 1993, Morgan Kaufmann Pub.
17. A. Helsinger, A. Thome and T. Wright. Cougaar: A Scalable, Distributed Muti-Agent Architecture. In Proc. IEEE Conf. on Systems, Man and Cybernetics (SMC), 2004.
18. A. Krishna, D. C. Schmidt and R. Klefstad, Enhancing Real-Time CORBA via Real-Time Java. In Proc. 24th IEEE Int'l Conf. on Distributed Computing Systems (ICDCS), 2004.
19. N. Mehta, N. Medvidovic and S. Phadke, Towards a Taxonomy of Software Connectors. In Proc. Int'l Conf. on Software Engineering, 178-187, 2000.
20. J. Miller and J. Mukerji (Eds.), Model Driven Architecture (MDA). http://www.omg.org/cgi-bin/doc?ormsc/2001-07-01, 2001.
21. M. Portmann, S. Ardon, P. Senac, A. Seneviratne, PROST: A Programmable Structured Peer-to-peer Overlay Network, In Proc. IEEE Int'l Conf. on Peer-to-peer Computing (P2P), 2004.
22. Y. Smaragdakis and D. Batory, Mixin Layers: An Object-Oriented Implementation Technique for Refinements and Collaboration-Based Designs. Software Engineering and Methodology 11(2), 215-255, 2002.
23. B. Spitznagel and D. Garlan, A Compositional Approach for Constructing Connectors. In Proc. Working IEEE/IFIP Conf. on Software Architecture (WISCA), 2001.
24. S. Thakkar, C. A. Knoblock and J. L. Ambite, A View Integration Approach to Dynamic Composition of Web Services. In Proc. ICAPS Workshop on Planning for Web Services. 2003.
25. T. L. Veldhuizen and D. Gannon. Active libraries: Rethinking the roles of compilers and libraries. In Proc. SIAM Workshop on Object Oriented Methods for Inter-operable Scientific and Engineering Computing (OO'98), 1998.
26. D. M. Yellin and R. E. Strom, Interfaces, Protocols and the Semi-Automatic Construction of Software Adaptors. In Proc. Object-Oriented Programming, Systems, Languages and Architectures (OOPSLA), 176-190, 1994.
27. R. Meier, Taxonomy of Distributed Event-Based Programming Systems, 1st Int'l Workshop on Event-Based Systems (DEBS 2002), July 2002, Vienna, Austria.
28. P. Th. Eugster et al., The Many Faces of Publish/Subscribe, ACM Computing Surveys, Vol. 35, Issue 2, June 2003.

Optimizing the Access to Read-Only Data in Grid Computing

Alek Opitz and Hartmut Koenig

Computer Science Department,
Brandenburg University of Technology Cottbus,
Germany
{ao, koenig}@informatik.tu-cottbus.de

Abstract. A fundamental problem of grid computing is the communication overhead. One reason of this overhead is the access to remotely stored data. Caching read-only data is a possible alleviation of the problem. In case of grid computing caching can be optimized by using allocation schemes considering the contents of the caches. Possible ways to achieve such an allocation in a grid are the topic of this paper. The paper proposes to use allocation schemes preferring resources with the required data in their caches. In doing so the hit rate of the caches will be increased and as a consequence the average response time of the jobs and the network load will be reduced. Two new possible allocation approaches are discussed and compared with classical allocation schemes. The performance and the costs of the schemes (when applied to large grids) are evaluated using a simulation environment.

1 Introduction

In recent years strong efforts have been made to use idle computing resources distributed over the Internet. The idea of grid computing was born. It aims at giving users with compute-intensive applications the possibility to submit their jobs to a grid that provides the required resources. A fundamental problem of this approach is the access to the data. The communication overhead makes a lot of jobs unsuitable for grid computing. Therefore it is a quite obvious goal to reduce this communication overhead.

In this paper we focus on optimizing the access to read-only data.[1] Such immutable data play an important role in grid applications ([11], [1]). A typical example is virtual screening([32]), which is used in the development of drugs for medical treatments. Virtual screening analyzes chemical compounds described in databases. These databases are not specific for single applications and describe hundreds of thousands of compounds. Those compounds that do not show any activity against the target receptor (this is the majority) are excluded. Subsequent drug design phases only consider the remaining compounds. Because the analysis is a computationally complex task, it is desirable to analyze the compounds in parallel, which makes grid computing

[1] Note that in grid computing the program code is also a kind of read-only data that has to be transferred to the executing resources.

very reasonable. Other grid applications which access read-only data are, for example, the rendering of animation movies, where the program code and the model data must be accessed ([28]), and applications accessing the protein data base([5], [30], [3]).

Since data often do not exclusively belong to a certain application, it is possible that different jobs access the same data, e.g. the same databases or the same program libraries. To accelerate the access to the data caches might be useful. Caches exploit the fact that many files are accessed multiple times. They store the recently used files in the hope that some of these are required soon again. Usually it cannot be predicted, whether the data will really be reused. There is only a limited probability. However, in the case of grid computing this probability can be increased by preferably allocating the jobs to those resources that already have at least some of the needed data. Possible ways to achieve such an optimized allocation in a grid are the topic of this paper. We especially pay attention to the costs of the allocation when considering large numbers of resources.

The paper is organized as follows. In section 2 we discuss two possible allocation schemes for this problem. Section 3 describes the simulation environment used for evaluating the performance of the two schemes. The results of this evaluation are summarized in Section 4. In Section 5 we give an overview on related work. The final remarks give an outlook on future work.

2 Optimized Allocation According to Cache Contents

In the following discussion we consider a grid with resources provided, for example, by enterprises, universities or individuals. We further assume that there is a broker between the users and the resources which is responsible for allocating appropriate resources to the jobs. After allocation the jobs are executed and the results are finally returned to the users. As we focus in this paper on optimizing the access to read-only data stored elsewhere in the Internet and because we want to keep the model as simpleas possible, only serial jobs are considered. This is, of course, a simplification, but as discussed in Section 5, allocation mechanisms for parallel jobs concentrate on the parallel allocation of machines. This type of optimization is largely orthogonal to the optimizations discussed in this paper. The scenario is shown in Fig. 1.[2]

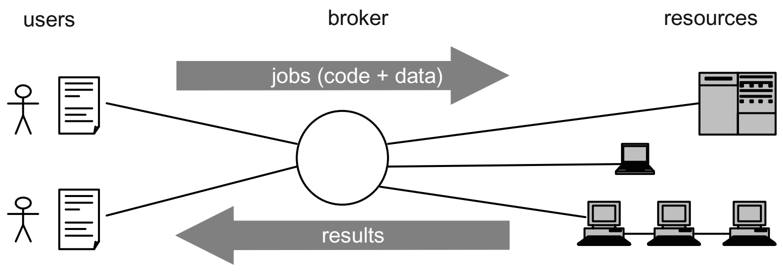


Fig. 1. Assumed scenario

[2] Note that the broker is not necessarily involved in the transport of the data. It is possible that the resources load the data directly from the sources.

In order to allocate resources that have the needed data locally available, we use directory servers. These servers store information about the locations of the existent replicates. According to the terminology used in [11] the servers are called *RLIs* (*replica location indices*). They receive their information from *local replica catalogs* (LRCs*)*. An LRC belongs to a single place and contains information about the data that are locally available to the resources at this place.

Since allocation schemes for grid computing are considered, attention has to be paid to the scalability problem, i.e. the solution must also work efficiently for large numbers of resources and jobs. For that reason, an infrastructure with several RLIs seems more appropriate for managing replicates than a single directory server. Obviously there are (at least) two possibilities for portioning the replicate management data. The first one is to partition the resources and to make each RLI responsible for such a partition. This possibility is discussed in Section 2.1. An alternative approach is to partition the data objects and to make each RLI responsible for one partition. This approach is considered in Section 2.2.

2.1 RLIs with Distinct Sets of Resources

The basic idea of this approach is that the individual RLIs only store information about distinct subsets of LRCs. Hence a single RLI can only return resources from a subset of the available resources. In order to avoid this, a hierarchical arrangement of the RLIs is proposed (see Fig. 2). The search for a resource with a certain data item starts at the top level RLI. This RLI (and any else) contains only condensed information about the locations of the data items, i.e. an RLI knows for each child only the set of data items that are available at any resource being descendant of this child node. Thus it is even possible to search for a resource with several needed data items instead of searching for a resource with a single data item. Obviously in this case the hierarchical approach does not guarantee to find the optimal resource, i.e. the resource with the most of the desired data items. On the other hand, the space requirements per server are reduced and the allocation process is made faster.

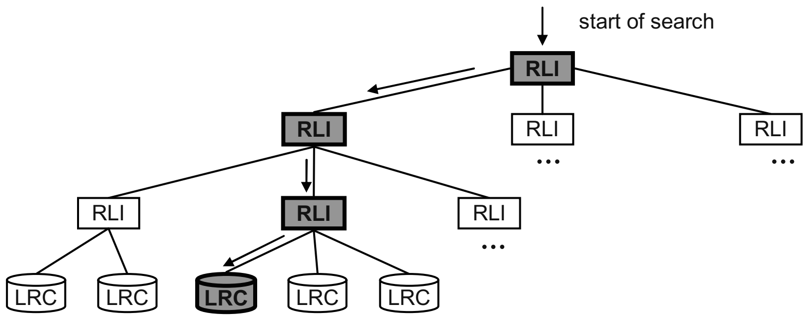


Fig. 2. Hierarchical selection of resources

Using Bloom Filters. The described hierarchical structure reduces the space requirements on the individual servers. An additional reduction seems possible by using a lossy compression of the information about the replicates. For this purpose, the use of Bloom filters was proposed ([7], [11]). Bloom filters are able to store information about the availability of data objects in an efficient way. This is achieved by applying several hash functions to a single data object. The hash functions are usually independent, but they all map onto the same domain, e.g. the range 1...*l*., i.e. onto a bit string of length *l*. When new data objects are stored at a resource, the bits calculated by the hash functions are set to 1.[3] To test the availability of a certain data item the corresponding bit positions have to be checked. Obviously this can lead to the erroneous assumptions about the availability of data items, since the same bits can be set by other items, too.

Several Top Level RLIs. Since the top level node receives all the queries, the model depicted in Fig. 2 is neither scalable nor reliable. This problem can be alleviated by introducing some additional access points. Preferably each resource should be reachable from each access point. Several topologies are conceivable, especially topologies of space division switches ([29]). For our approach, we use a simple model which is depicted in Fig. 3. Each resource belongs to exactly one of several distinct trees. The roots of these trees notify all the access points about the available data (and possibly further information). The result is a topology in which each resource can be reached from each access point on exactly one path.

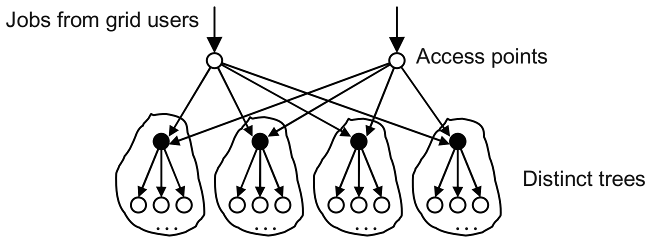


Fig. 3. Model with several access points

Using Additional Information. Finally the question arises, whether the allocation of resources should only depend on the required data objects. Our simulations showed that this is inefficient, because knowledge about the current load of the resources is extremely important. Therefore a hierarchical load balancing was added, i.e. the resources notify the RLIs not only about the available data items but also about their current load. Thus an RLI can select the appropriate child taking both the load and the available data items into account.

Costs of the Approach. In order to assess the benefit of the approach, the additional overhead of the proposed allocation scheme has to be compared with its advantages.

[3] To allow the removal of data items, a counter is needed for each bit position. These counters are only required at in the LRCs, not in the RLIs.

Possible advantages are a reduced average response time, a reduced network load, and a reduced load at the data sources. The additional costs can be divided in:

(1) costs for the notification of the load and available data items at the resources
  - (1a) number of bytes sent,
  - (1b) costs at the resources (time busy with sending notifications),
  - (1c) costs at the directories (time busy with receiving notifications)

(2) costs for selecting suitable resources
  - (2a) number of bytes sent,
  - (2b) costs at the directories for selecting the resources.

We now discuss the individual parts of these costs. The simulation results are presented in Section 4.

Costs for Notifying the Directory. First the messages sent from the resources to the directory servers are considered. The indication of available data is mapped into Bloom tables with n bits. The load information is very small, e.g. 4 bytes. Both data are sent from the children nodes to their fathers. An overhead of h bytes is added to each message. In our simulations we used a value of $h = 50$ bytes derived from the size of the IP-header and additional overheads for other protocols. Different values of h did not greatly influence the results.

The hosts sending and receiving these messages have only a small computational overhead for handling these messages. Therefore the busy times are approximated by the serialization delays, as it can be assumed that NIC and CPU work in parallel.

Costs for Selecting a Resource. To select suitable resources the RLIs must be queried. A query contains a characterization of the set of the needed read-only data plus an overhead of h bytes. As the directory servers store only the Bloom tables it is sensible to send only the hash values of the data objects. The necessary size of the hash values can be derived from the size of the used Bloom tables. Such a query is sent to an access point and from there down through the hierarchy. Finally the answer containing the selected resource is returned to the source of the query.

Each queried RLI has to select the most appropriate child. It has to compare the current load and the Bloom tables of the children. As only the best child node has to be identified, a linear search through the children is sufficient. Usually there are only few children, so the selection of the most appropriate child is not very complex. Therefore the serialization (and deserialization) delays of the messages are equated with the aggregated costs at the involved RLIs.

2.2 RLIs with Distinct Sets of Data Items

We now consider the second approach in which the servers contain information about distinct sets of data items. The structure is depicted in Fig. 4. In this solution the amount of data per RLI can be kept constant by increasing the number of RLIs proportionally to the number of replicates. Of course there must be an efficient mechanism for identifying the RLI which contains the information about a certain data item. This can be done by means of hash functions, in case of varying sets of RLIs consistent hash functions ([24]).

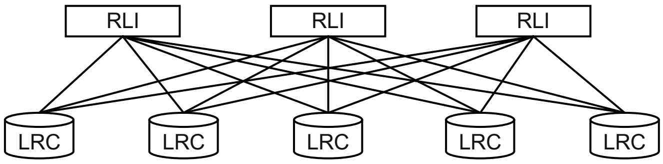


Fig. 4. Partitioning of the data sets

Selecting Suitable Resources. A problem of the depicted scenario is that a job might require several data items. However, a single RLI knows only the locations of data items from a limited subset. That means, in order to find the optimal resource for a job with multiple needed data objects it were necessary to query several RLIs for resources with the needed items. Finally the resource lists returned by the RLIs had to be unified. Sending and unifying these lists is potentially expensive. Therefore in this paper we do not follow this approach. Instead we use a mechanism that searches for a resource with the most important (i.e. biggest) of the needed data items. We regard this approach as a first step, which delivers a lower bound for the effectiveness of the method. An examination of possibilities considering several data items is planned for the future.

Using Additional Information. For the same reasons as in Section 2.1, load balancing is added. A load server is introduced that informs about the load of the available resources. The resources have to update these data when getting idle or loaded. The combined approach actually searches for resources that are unloaded and have the most important data item locally available. If no such resource can be found, any unloaded resource will be used.

Costs of the Approach. The costs of this approach are analyzed analogously to Section 2.1.

Costs for Notifying the Information Servers. The information about available data items is sent from the LRCs to the RLIs. To identify the data items URLs with an average length of $u = 100$ bytes are used. Alternatively hash values could be used what is not considered in this paper. The load information messages are sent to the load servers. It is sufficient to update the server only when the load changes. As in Section 2.1 an overhead of h bytes per message is added and the busy times of the involved hosts are approximated by the serialization delays.

Costs of Selecting Resources. To select a suitable resource a request is sent to the RLI responsible for the most important (largest) data item. The request contains the item's identifier which is assumed to be a URL with an average length of u bytes. The RLI finds the resources with the required item and sends the list to the load information service where one of the resources is selected. The response with the selected resource is returned to the requesting host. For each message, an overhead of h bytes is added. Analogously to the other estimations the time for processing the messages is approximated by the (de)serialization delays.

Optimization. A simple optimization of this allocation method is to ignore small data items. For the simulations described below a limit of 1 MB was assumed, i.e. resources with a size below 1 MB are not indicated to RLIs. Consequently, if the largest read-only data item needed by a job is smaller than 1 MB, any unloaded resource is selected.

3 Simulation

To assess the benefit of the proposed mechanisms we performed extensive simulations. The underlying model is explained in this section.

3.1 Grid Model

The environment developed for these simulations was kept as simple as possible. Thus it was possible to simulate even large grids on desktop PCs. The model consists of a configurable number of computational resources which are modeled as single-processor machines. As only serial jobs are considered this should be no problem. In the standard configuration 4096 resources were simulated. It was assumed that they have all the same properties, e.g. the same speed. Failures in the grid are not considered. Each machine is connected to the Internet and has its own cache for storing data. A cache stores each needed (read-only) data object. In case of shortness of space the least recently used objects will be removed from the cache. As at the beginning of the simulation the caches are empty, we started the measurements only after simulating the grid for a certain amount of time.

Usually an Internet connection is shared by machines of the same organization. Thus actually a single connection (with a relatively high data rate) for a set of machines should be modeled. For simplicity reasons and thus enabling simulations of large grids such shared connections were not modeled. Instead a separate link with a lower bandwidth was assumed for each machine. This model rather corresponds to a grid with privately used PCs than to a grid using the PCs of enterprises. The consequences of this simplification still has to be analyzed what is planned for the future.

Besides the resources there is the infrastructure for distributing the jobs. For simplicity, the machines of this infrastructure are connected to the Internet with the same low bandwidth as the resources. This might underestimate the available data rate, but actually this data rate is mainly used for approximating the load of the machines of the infrastructure (see Section 2.1). An increased network bandwidth does not increase the computational speed of the machines.

3.2 Workload Model

Besides modeling the resources it is at least equally important to model the workload properly. For this, a special library for generating jobs was developed. The parameters of a job are the pure execution time and additionally the required read-only data items. Each item is characterized by the pair (identifier;size). It is assumed that the needed data items are loaded completely into the cache of the executing resource.

The information for modeling the workload has been taken from several papers about workloads. The papers belong to the areas of grid computing, parallel machines and web caching. Because it is not sure that the found properties taken from other research really applies to the grid computing as discussed, quite a few variations in the workload have been tested for validating the results (see Section 3).

Inter-Arrival Times. According to [12] and [23] the distribution of inter-arrival times can be modeled by a hyper-exponential distribution, i.e.:

$$F(x) = p_1 \cdot (1 - e^{-\lambda_1 x}) + (1 - p_1) \cdot (1 - e^{-\lambda_2 x})$$

Variations due to daytime as reported, for example, in [9] are ignored. In the simulation the parameter values $!_1 = 0.000204$, $!_2 = 0.0038$ and $p_1 = 3.46\,\%$ were used ([23]). For scaling the load of the grid the parameters, $!_1$ and $!_2$ are multiplied by a factor *f*. Thus the mean value μ of the time between two successive job submissions is reduced by the factor *f*, whereas the coefficient of variation (*CV*) stays constant.

Runtimes. Different models exist for the distribution of job runtimes. [23] proposes a hyper-exponential distribution, [14] and [20] suggest a log-uniform distribution, [12] and [16] use a Weibull distribution. Fortunately, the models are very similar. In our simulation a log-uniform distribution was used. Additional tests with the Weibull distribution were made. The distribution function of a log-uniform distribution can be expressed by the following formula:

$$F(x) = a + b \cdot \ln x$$

The values of *a* and *b* are very similar in [14] und [20], for the simulation the values $a = -0.24$ and $b = 0.111$ were used.

Popularity of Data Items. For the files of the WWW, it has been shown that the popularity of the individual files follows a Zipf-like distribution ([8], [27], [4]):

$$P(i) = \frac{''}{i^{\gamma}} \quad \text{with} \quad 0 < \gamma \leq 1 \quad \text{and} \quad '' = \overset{\&N}{\underset{\%i=1}{}} \frac{1}{i^{\gamma}} \overset{\#}{!}^{-1}$$

According to the literature the value of parameter # is somewhere between 0.6 und 0.9, in our simulations # = 0.7 was assumed. Besides parameter # the number of different files is important. For our simulations we selected this number in such a way that the used data items were accessed on average 3.6 times during the simulation which is in line with values reported in literature ([18], [27], [10], [22]).

Because it is unclear, whether this Zipf-like popularity distribution can be applied to files in a grid environment, additional tests with other parameter values and with a uniform distribution were carried out.

Needed Read-Only Files. To simulate the delays to access read-only files their sizes must be modeled. The distribution of files sizes is usually not uniform. Instead [15] proposes a log-uniform distribution:

$$F(x) = \Phi \overset{\&}{\underset{\%}{}} \frac{\ln x - \mu'}{\sigma'} \overset{\#}{!} \quad \text{with} \quad \sigma' > 0$$

Following the studies presented in [31] the values *$'* = 0.01826 and μ' = 8.699 were chosen for the standard configuration. Besides the sizes of single files the number of files per job is important. As no empirical data was found for this number, the simulations were conducted with very different ranges of this number. In the standard configuration a uniform distribution over the interval [1;5] was chosen.

4 Simulation Results

The results of our simulations are presented in Table 1. We simulated five allocation schemes which are explained below. The second column shows the average load of the grid resources. Further, the average response time (*ART*) was measured. This is the time between the submission of a job and the return of the results. The overhead of a job is the time waiting in a job queue plus the time for loading the read-only data from a remote location. It also includes the time for notifying the used allocator. The overhead and the ART are given in seconds. The column “kbps/resource” indicates the network load. For conceivability reasons, the aggregated network load is divided by the number of resources. The rightmost column shows the demands on the infrastructure. It gives the number of needed machines and is determined by the total busy time divided by the whole simulated time. The demands are only estimates and are quite low for the simulated number of resources and for the used average job length. But increasing the number of grid resources and decreasing the average job length increases the demands. Thus the column gives an indication of the relation among the different allocation schemes.

Table 1. Results of the allocation schemes

Allocation scheme	Load	ART	Overhead	kbps/resource	Nodes
Random	69.7 %	49037	41134	12.1	0.00000
Round Robin	69.7 %	36415	28495	12.1	0.00000
Load Balancing	69.5 %	8053	137	12.0	0.00074
Resource partitioning (see sect. 2.1)	69.5 %	8052	137	11.9	0.02005
Data partitioning (see sect. 2.2)	69.3 %	8026	111	9.6	0.00410

The first three allocation schemes of Table 1 were introduced for comparison. The first scheme selects the resources randomly, the second one in a round robin fashion. The third allocation scheme named “Load Balancing” uses a load directory as mentioned in section 2.2 to select unloaded resources. The results clearly show the benefit of taking the resource load into account. For that reason, we use the “Load Balancing”-allocator as reference for our two allocation schemes which are shown in the last two rows.

As it can be seen from the table, the approach with RLIs responsible for distinct partitions of the resources (see section 2.1) cannot achieve the desired optimization (in relation to the reference allocation scheme). The main reason is the hierarchical selection of resources, which leads to unfavorable decisions in the process of resource allocation. In contrast the second approach (with RLIs responsible for distinct partitions of the data items, see section 2.2) indeed helps to significantly reduce the overhead and network load.

Due to limited space we present in Table 1 only the results of the standard configuration (as described in Section 3). However, simulations with various parameters showed that the relation between the five allocation schemes is largely independent of the exact values of most parameters, as, for example, the overhead per message or the size of the grid. The most significant parameters are the network bandwidth and the sizes of the data items. That means the ratio between the overheads of the individual approaches remains largely constant. As the reduction of the ART directly relates to the reduction of the overhead, it is clear that the relative reduction of the ART depends mainly on the ratio between the overhead and the ART. If this ratio is increased, the relative reduction is also increased. Correspondingly, if the ratio is decreased, the relative reduction is decreased, too. It is not quite clear, whether the ratio of our standard configuration is realistic in respect of this point. Therefore further investigations are needed.

5 Related Work

To the best of our knowledge no papers exist on resource allocation schemes as proposed in this paper. However there are other proposals with at least partially the same goal, i.e. optimizing the access to data or optimizing the resource allocation. Hence in this section a brief overview of these proposals is given. First we discuss proposals related to caching. Thereafter efficient methods for accessing remote data are reviewed and then we take a look at allocation schemes for parallel systems. Finally we discuss the applicability of methods used in the area of distributed databases.

In the WWW caching is used extensively to optimize the access to read-only data. Karger et al., for example, discuss the arrangement of the Web caches [24]. Instead of using a single hierarchy they propose to use individual hierarchies for different data objects. The trees have to be derived from the data identifiers by using hash functions. Because the set of participating caches is variable, a consistent hashing is used for reducing the costs of adding or removing a cache. Caching may also be applied to grid computing. In [2] a data grid is described that enables to access the same data from all over the world. To achieve a better efficiency the data are cached. GASS [6] is a data movement and access service for wide area computing systems which is optimized for grid applications. It exploits the fact that strong consistency among replicates is usually not necessary. Consequently the service uses caching for both reading and writing of the data.

Caching can increase the probability that a certain data item is locally available. If it is not locally available, it has to be retrieved from a suitable source. To do this in an efficient manner, two things are required [1]: an appropriate protocol for the transmission of the data and an efficient management of the copies (replicates). For the latter, a directory can be used containing for each data object the locations where it is stored. As argued in [1] users prefer working with groups of files instead with individual files. Therefore directories should contain the locations of data collections instead of locations of single files. Chervenak et al. also propose the usage of directory servers [11]. The paper describes a framework for the construction of services for localizing replicates. It is rather a classification of possibilities than a tangible concept for such a service.

Besides work focussing on data access there are a lot of papers about scheduling and resource allocation. A survey of such methods for parallel machines is given in [19]. The methods concentrate on the allocation of processors for jobs that need multiple processors at the same time. These methods are mainly orthogonal to the idea followed up in this paper. The same applies to [33] which proposes an allocation scheme that groups the processors into pools with fast connections inside the pools. Thus a parallel job should be preferably allocated to resources from a single pool. Fewer papers exist on the allocation of machines in the context of grid computing. In [21] it is emphasized that centralized methods are not suitable and some mechanisms are compared. An economical approach has been proposed in [17]. The individual jobs have utility values, the machines have machine values (i.e. costs) and the jobs are assigned to machines in an auction-like manner. [20] shows that due to the separation of local and global job queues an efficient global allocator needs regular notifications about the current state of the resources.

In the context of distributed databases the problem of finding suitable places for executing jobs arises, too. The approaches usually focus on the execution of the parts of a request at the place of the data – as far as this is possible. Only the unification is done on a coordinator ([13]). Such an approach is much more difficult in case of grid computing. At first general purpose programs cannot be decomposed as simple as SQL queries. SQL queries have a certain structure that delimits the computational power, but greatly alleviates the analysis and decomposition. Furthermore the most important idea of grid computing is the utilization of otherwise idle resources. However a location having the data locally available does not necessarily possess the needed computational resources.

6 Conclusions and Future Work

The paper discussed a possibility for optimizing the access to read-only data in case of grid computing. It presented the idea of increasing the cache hit rate by using allocation schemes regarding the contents of the caches. In order to proof the usefulness of the idea two possible allocation schemes were presented and evaluated by simulation. The calculation made by the developed simulation environment included also the costs of the allocation scheme – something that is hardly done in other work on this topic. It was shown that one of the two allocation schemes (the one with RLIs for partitions of data items, see section 2.2) indeed helps increasing the hit rate and thus reduces the overhead as well as the network load. This achievement is especially remarkable when considering that the simulated allocation scheme has not been optimized, yet. For example, the demands to the infrastructure might be reduced by using hash values instead of URLs for the data items. Furthermore, considering multiple data items instead of only the largest one could improve the effectiveness of the allocation scheme. The integration of these optimizations is planned for future work.

Even though we tried to follow the reality by simulating the execution of the jobs, it is desirable to verify the results in more realistic simulations. Especially failures in the grid and the heterogeneity should be considered what has not been done in this paper. As a starting point the results from [25], [26], and [18] could be used which discuss the generation of realistic grids.

References

1. Bill Allcock, Joe Bester, John Bresnahan, …: „Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing“ , IEEE Mass Storage Conference, 2001,
2. “Avaki Data Grid 3.0 Conceptual Overview”, Avaki Corporation December 2002, www.avaki.com
3. Kim Baldridge, Philip E. Bourne: “The new biology and the Grid”, "Grid Computing - Making the Global Infrastructure a Reality", edited by F. Berman, A. Hey, G. Fox, 2003 John Wiley & Sons
4. Paul Barford, Azer Bestavros, Adam Bradley, Mark Crovella: „Changes in Web Client Access Patterns. Characteristics and Caching Implications”, World Wide Web, Volume 2, Issue 1-2, 1999
5. Helen M. Berman, David S. Goodsell, Philip E. Bourne: „Protein Structures: From Famine to Feast”, AMERICAN SCIENTIST (July-August 2002), http://www.sdsc.edu/pb/papers/amer_sci.pdf
6. Joseph Bester, Ian Foster, Carl Kesselman, Jean Tedesco, Steven Tuecke: „GASS: A Data Movement and Access Service for Wide Area Computing Systems”, Sixth Workshop on I/O in Parallel and Distributed Systems, May 5, 1999
7. Burton H. Bloom: „Space/Time Trade-offs in Hash Coding with Allowable Errors“, Communications of the ACM, Volume 13, Number 7, July, 1970, pp. 422-426
8. Lee Breslau, Pei Cao, Li Fan, Graham Phillips, Scott Shenker: „Web Caching and Zipf-like Distributions: Evidence and Implications“, IEEE Infocom '99, pages 126-134, New York, NY, March, 1999
9. Maria Calzarossa, Giuseppe Serazzi: „A Characterization of the Variation in Time of Workload Arrival Patterns“, IEEE Transactions on Computers, 34(2): 156-162, 1985
10. Ludmila Cherkasova, Magnus Karlsson: „Dynamics and Evolution of Web Sites: Analysis, Metrics and Design Issue“, Sixth IEEE Symposium on Computers and Communications (ISCC'01), p.64, July 03-05, 2001
11. Ann Chervenak, Ewa Deelman, Ian Foster, …: „Giggle: A Framework for Constructing Scalable Replica Location Services“, IEEE Supercomputing 2002
12. Su-Hui Chiang, Mary K. Vernon: „Characteristics of a Large Shared Memory Production Workload”, 7th Workshop on Job Scheduling Strategies for Parallel Processing, Cambridge, MA, June 2001, Lecture Notes in Computer Science, Vol. 2221, Springer-Verlag

13. Peter Dadam: „Verteilte Datenbanken und Client-/Server-Systeme. Grundlagen, Konzepte und Realisierungsformen", Springer-Verlag, Berlin Heidelberg, 1996
14. Allen B. Downey, Dror G. Feitelson: „The Elusive Goal of Workload Characterization", Performance Evaluation Review 26(4), pp. 14-29, March 1999, http://www.cs.huji.ac.il/~feit/pub.html
15. Allen B. Downey: „The structural cause of file size distributions", Technical Report CSD-RT25-2000, Wellesley College, 2000, http://allendowney.com/research/filesize/
16. Darin England, Jon B. Weissman: „Costs and Benefits of Load Sharing in the Computational Grid", Workshop on Job Scheduling Strategies for Parallel Processing with Sigmetrics 2004
17. Carsten Ernemann, Volker Hamscher, Ramin Yahyapour: „Economic Scheduling in Grid Computing", from: D.G. Feitelson, L. Rudolph, U. Schwiegelshohn (Eds.): „Job Scheduling Strategies for Parallel Processing", 8th International Workshop, JSSPP 2002 Edinburgh, Scotland, UK, July 24, 2002, LNCS 2537, Springer-Verlag
18. Michalis Faloutsos, Petros Faloutsos, Christos Faloutsos: „On Power-Law Relationships of the Internet Topology", ACM SIGCOMM, Cambridge, MA, September 1999
19. Dror G. Feitelson, Larry Rudolph: „Job Scheduling in Multiprogrammed Parallel Systems. Condensed Version", Research Report RC 19790 (87657), IBM T. J. Watson Research Center, Oct 1994, Revised version from August 1997, http://www.cs.huji.ac.il/%7Efeit/pub.html
20. Jörn Gehring, Thomas Preiß: „Scheduling a Metacomputer With Uncooperative Sub-schedulers", from: „Job Scheduling Strategies for Parallel Processing", Springer Verlag, Editors: Dror G. Feitelson and Larry Rudolph, pages 179-201, 1999
21. Volker Hamscher, Uwe Schwiegelshohn, Achim Streit, Ramin Yahyapour: „Evaluation of Job-Scheduling Strategies for Grid Computing", 1st IEEE/ACM International Workshop on Grid Compuing, Springer Verlag, LNCS 1971, Bangalore, India, December 17, 2000
22. Adriana Iamnitchi, Matei Ripeanu: „Myth and Reality: Usage Behavior in a Large Data-Intensive Physics Project", SC2002, November 11-16, 2002, Baltimore, Maryland (poster), GriPhyN TR-2003-4
23. Joefon Jann, Pratap Pattnaik, Hubertus Franke, …: „Modeling of Workload in MPPs", from: "Job Scheduling Strategies for Parallel Processing", Springer Verlag, editors: Dror G. Feitelson and Larry Rudolph, pp. 95-116, 1997
24. David Karger, Eric Lehman, Tom Leighton, …: „Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web", Symposium on Theory of Computing, 1997
25. Yang-Suk Kee, Henri Casanova, Andrew Chien: „Realistic Modeling and Synthesis of Resources for Computational Grids", ACM Conference on High Performance Computing and Networking, SC2004, Pittsburgh , Pennsylvania, November 2004
26. Dong Lu, Peter A. Dinda: „Synthesizing Realistic Computational Grids", ACM/IEEE Supercomputing 2003 (SC 2003), November, 2003, Phoenix
27. Norifumi Nishikawa, Takafumi Hosokawa, …: „Memory-based architectur for distributed WWW caching proxy", 7th International Conference on World Wide Web, Brisbane, Australia, 1998
28. Pixar Animation Studios: „How We Make A Movie. Pixar's Animation Process", http://www.pixar.com/howwedoit/index.html, Download: 7.6.2004
29. Martin de Prycker: „Asynchronous Transfer Mode", Prentice Hall, 1996
30. Volker Strumpen: „Volunteer Computing", Software - Practice and Experience, Vol. 25(3), 291–304, März 1995

31. Feng Wang, Qin Xin, Bo Hong, Scott A. Brandt, Ethan L. Miller, Darell D. E. Long, Tyce T. McLarty: „Large-scale virtual screening for discovering leads in the postgenomic era“, 21st IEEE / 12th NASA Goddard Conference on Mass Storage Systems and Technologies, pages 139–152, College Park, MD, April 2004
32. B. Waszkowycz, T. D. J. Perkins, R. A. Sykes, J. Li: „Large-scale virtual screening for discovering leads in the postgenomic era“, IBM Systems Journal, Volume 40, Number 2, 2001, ("Deep computing for the life sciences"), http://www.research.ibm.com/journal/sj/402/waszkowycz.html
33. Songnian Zhou, Timothy Brecht: „Processor Pool-Based Scheduling for Large-Scale NUMA Multiprocessors”, 1991 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, San Diego, California, USA, 21st-24th May 1991

Using Data Item Relationships to Adaptively Select Data for Synchronization

Oskari Koskimies

Nokia Research Center, Itämerenkatu 11-13, 00180 Helsinki, Finland
oskari.koskimies@nokia.com

Abstract. Though synchronization of email and PIM data to mobile devices has been a major practical concern for a long time, there has been relatively little advance in making synchronization work adaptively. We examine in this article the possibility to adapt synchronization to bandwidth and resource constraints by only synchronizing the items that are currently relevant to the user, and present initial results suggesting that item relationships are helpful for accomplishing this task.

1 Introduction

Adaptive synchronization has been studied at Nokia Research Center in connection with SyncML [1]. Instead of the traditional approach of protocol and content optimization, the idea is to intelligently partition the user's data between terminal and network, choosing what to synchronize and when so as to maximize efficiency. The task is complicated by the unreliable nature of wireless connections – if user data resides in the network, access to it will be slow when network conditions are poor, and impossible when in disconnected state. It is therefore imperative to ensure that data important to the user is available locally on the mobile terminal.

Selection of data must take into account usage context. For example, the data needed on a business trip is usually quite different from that required on a holiday. Synchronization should also adapt to the network, allowing more data to be synchronized when network conditions are favorable, and to the terminal, storing more files locally if there is ample space on the terminal.

2 Related Work

When examining prioritizing synchronization systems, the venerable CODA system, presented by Kistler and Satyanarayanan e.g. in [2], must of course be mentioned. To facilitate disconnected operation, the CODA distributed file system attempts to ensure that critical files are available in the cache. The process utilizes both access history as well as explicit configuration information provided by the user.

General-purpose email classifiers used to identify spam [3][4] could be applied to our problem as well. Instead of looking for email similar to known spam, the classifier would look for email similar to emails known to be important (e.g. by observing user email reading behavior). The problem with this approach is that the importance of emails depends on current tasks and may change on a daily basis; traditional classifiers do not adapt fast enough.

The closest related work is actually related to email prioritization. The main difference in this case is that when prioritizing email for presentation, you only care about the unread emails. This is reasonable since the user knows about the already read emails and can easily view them if desired. In synchronization, however, also read emails are important, since they may contain information that the user needs while mobile. For example, an old email about a soon-to-be meeting might contain instructions for driving to the meeting venue. Examples of static rule systems include CLUES [5] and Bifrost [6]. CLUES uses information from a user's work environment (calendar entries, sent emails, etc.) to extract clues about his short term-interests for the purpose of prioritizing the messages. The Bifrost system organizes a user's inbox into groups (messages related to calendar events, personal messages, etc.), which enables the user to more easily spot important mails. PRIORITIES [7], on the other hand, is a learning system which classifies emails according to criticality based on sender identity and relation to user, number of recipients, textual content, etc.

3 Adaptive Synchronization

Adaptive synchronization provides a way for autonomously determining what are the most important data items for synchronization in a given context. The algorithm utilizes application-specific a priori knowledge about data items the user is certainly going to need (e.g. new mails) and follows relationship paths between data items to find other data items related to the needed items. The underlying basic assumption is that if the user needs a certain data item, then he is also likely to need related data items. The use of relationships is necessary to identify data items that may be quite old and of little apparent importance by themselves, but are relevant to the user because they are closely related to other, currently important items.

The idea behind our approach is that relationships between data items describe the probability that if the user needed the first item, he will also need the second item. For example, the relationship between an email and an attachment of the email might have the weight 0.9, signifying that if the user needed the email, with 90% probability he will also require the attachment. We refer to this weight as the *relevance* of a relationship. The *utility* of an item quantifies the benefit of the item to the user.

An initial set of data items, which are certain to be needed by the user, is assumed to be known (for example, new mail and today's calendar entries). Determining the initial set is application-specific. The expected utility (overall importance) of an item depends then on both its utility and on the relevance of the relationship path(s) to it from the initial set, which represents the probability that the user will in fact need the item. The expected utility of an item can be penalized in the final calculation with the size of the item, so that large data items are synchronized only if they have very high expected utility (optimizing "utility per kilobyte").

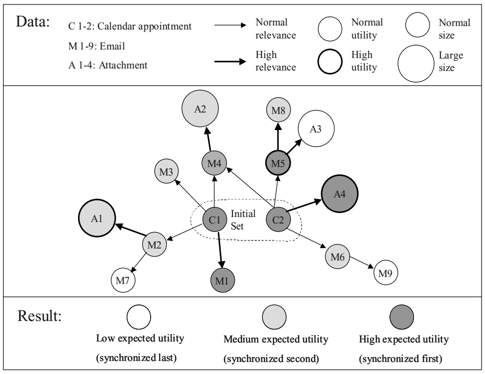


Fig. 1. Adaptive synchronization example

In Figure 1, an example relationship graph is shown. The figure also shows an overview of relevancies, utilities, item sizes and calculated expected utilities. Note that in general, the relationship graph will contain loops. However, as it is our purpose to explain the general idea, the handling of loops is an unnecessary complication.

Calendar appointments C1 and C2 in Figure 1 make up the initial set. All relationship paths are calculated from that set. The email messages M1-M6 are from people participating in one of the meetings, the sender of M4 participates in both. The email M5 has high utility because the sender had set the high importance flag on. Email M7 is an earlier mail from the same sender as M2, similarly for M9 and M6. Email M8, on the other hand, is an email from both the same thread and sender as M5. Attachments A1, A2 and A3 belong to emails M2, M4 and M5 respectively, and attachment A4 belongs to calendar appointment C2. The attachments are much larger than emails. Attachments A1 and A4 have high utility because they are PowerPoint files which can be viewed on a mobile terminal relatively efficiently, unlike A2 and A3 which are MSWord documents. Relationships M2A1, M4A2, M5A3 and C2A4 have high relevance since an attachment is usually highly relevant to the containing email/appointment. Relationship C1M1 has high relevance because the organizer of the meeting has sent M1. Relationship M5M8 has high relevance because M8 has both same sender and thread as M5.

The approach takes context into account in two ways. Firstly, the initial set contains currently relevant items (e.g. recent mails and appointments), which causes the algorithm to automatically select items related to the user's current context. Secondly, relationship and utility weights can be adjusted based on context. For example, if the user expects to have low-bandwidth connectivity available continuously, utility of large items might be increased since small items can be fetched on-demand using the low-bandwidth connection. Similarly, the age of an item might affect its utility.

4 Interim Results

To evaluate the correctness of our assumption that being related to another important item is a good predictor for item importance, we need to have a priori knowledge of items the user actually requires next. We are implementing a tracker application for Microsoft Outlook, which enables us to compare the items selected by the algorithm to the items that were actually required by the user. In the meantime, we have done some initial tests using a simple HTML form generated from the contents of the user's Microsoft Outlook data. The users were asked to assume that they were about to go on a one-week business trip, and then grade in that context the importance of each item on a scale from 1 to 5 using the form. The rather laborious form-filling aspect of the test setup prevented the gathering of a statistically significant set of data; however, the initial results are already encouraging, and when the Outlook Tracker application is finished, collection of user data will be much simpler.

With user-provided importance data available, we tested the hypothesis for each relationship type R and item type A by examining the probability that type A items, targeted by a type R relationship originating from an important item, were important only randomly. The probability was calculated using the Hypergeometric distribution. In testing with two large email sets from two different users, we got the results shown in Table 1. If the probability P("importance is random"), denoted as P(rand), is very small (<0.05), then we can with high confidence say that type R relationships are indeed good predictors for importance. The entries that signify over 95% confidence

Table 1. Relationship predictive power

Relationship	Source item type	Target item type	P (rand) Dataset1	P (rand) Dataset2
Previous email by same sender	Email	Email	**0.00046**	**0.02095**
Next email by same sender	Email	Email	**0.00001**	**0.00578**
Last email from same sender	Email	Email	**0.00071**	0.10272
Email from organizer	Calendar entry	Email	0.05029	0.86095
Email from required participant	Calendar entry	Email	**0.02661**	**0.04725**
Contact information for sender	Email	Contact information	**0.00000**	**0.00000**
Contact information for receiver	Email	Contact information	**0.00000**	**0.00068**
Contact information for organizer	Calendar entry	Contact information	**0.00000**	0.09370
Contact information for required participant	Calendar entry	Contact information	**0.00055**	**0.00109**

in the predictive power of the relationship are in bold. For the most part, the results are quite intuitive, but it is surprising that emails from meeting participants are more likely to be important than emails from the meeting organizer. This is likely because the organizer usually sends information related to the meeting as updates to the calendar entry, which are processed by Microsoft Outlook into the original calendar entry and are thus not visible as independent email entries. Other participants, on the other hand, have to rely on normal email for issues related to the meeting.

Note that there are no relationships in the above table that would point to calendar entries. Several such relationships were in fact examined, but none were good indicators, with even the best one (meeting participant is email sender) having a value of around 0.38 for dataset 1, i.e. a 38% probability of having no predictor value. We conclude that while relationships *from* calendar entries make good predictors, the same is not true for relationships *to* calendar entries. In other words, calendar entries are very good candidates for the initial set.

The selection accuracy of the algorithm was slightly above 90% for the two datasets; reasonable considering that no machine learning was used, but not yet good enough for practical use. The percentage of important items was 13% for Dataset 1 and 26% for Dataset 2.

5 Summary and Future Work

We have developed a relationship-based approach for autonomously selecting emails for synchronization based on their importance. The interim results indicate that relationships are good predictors for email importance, but the algorithm would need machine-learning features to perform well enough. We plan to combine the relationship-based approach with a traditional classifier approach to achieve this.

We are in the process of instrumenting Microsoft Outlook so that it records user activity such as the order and speed in which emails are read. From this information, an importance classification will then be extracted and used to more rigorously test our hypothesis of item importance and to evaluate the performance of the selection algorithm. The Outlook Tracker will also provide us a method for gathering data from which the selection algorithm can learn and adapt to user behavior.

References

[1] Open Mobile Alliance: "SyncML Data Synchronization Specifications", Version 1.1. Available electronically from http://www.openmobilealliance.org/tech/affiliates/syncml/syncmlindex.html.

[2] Kistler J. J. and Satyanarayanan M.: "Disconnected Operation in the Coda File System". ACM Transactions on Computer Systems, Vol. 10, No. 1, pages 3-25, February 1992.

[3] Graham, P.: "A Plan for Spam", August 2002. Available electronically from http://www.paulgraham.com/spam.html

[4] Yerazunis W.: "Sparse Binary Polynomial Hashing and the CRM114 Discriminator", in Proceedings of the 2003 Spam Conference, January 2003. Available electronically from http://spamconference.org/proceedings2003.html.

[5] Marx M. and Schmandt C.: "CLUES: Dynamic Personalized Message Filtering", in Proceedings of CSCW `96, pages 113-121, November 1996.

[6] Bälter O. and Sidner C. L.: “Bifrost Inbox Organizer: Giving users control over the inbox”, in Proceedings of the Second Nordic Conference on Human-computer Interaction, pages 111-118, October 2002.
[7] Horvitz E., Jacobs A. and Hovel D.: “Attention-Sensitive Alerting”, in Proceedings of the 15th Conference on Uncertainty and Artificial Intelligence, pages 305-313, July 1999.

A Client/Intercept Based System for Optimizing Wireless Access to Web Services

Irene Kilanioti, Georgia Sotiropoulou, and Stathes Hadjiefthymiades

University of Athens, Department of Informatics and Telecommunications,
Panepistimioupolis, Ilissia, Athens 15784, Greece
{grad0553, gsot, shadj}@di.uoa.gr

Abstract. In this paper we discuss the introduction of Web Services in the wireless/mobile domain. The use of Web Services is gradually expanding into the mobile internet area where the population of users is rapidly increasing. Web Services offer standardized ways of creating, publishing, searching and invoking services and provide a very important platform for the development of mobile e-commerce. We identify problems related to the use of the verbose protocols of Web Services (i.e., SOAP/HTTP) over the "expensive" wireless medium. Our architecture assumes the existence of WS-aware software (client or provider) in the wireless device. We try to optimize the HTTP/SOAP stream exchanged over the wireless medium. Our effort is largely based on the IBM WebExpress design. Our measurements indicate substantial benefits for the users.

1 Introduction

The extraordinary growth evident in the area of wireless and mobile technologies creates a dynamic environment that produces diverse wireless technologies and standards, in contrast to other areas of communications marked by a convergence toward uniformity. All this activity will bring the reality of the coming decades close to the vision of "anytime, anywhere" communications.

With the simultaneous exponential growth of the Internet, wireless users are now seeking Internet capabilities equivalent to those provided by a fixed network. Specifically, the growth of wireless telecommunications stimulated the interest for the so-called "nomadic computing"[1] [1], which aims to provide users with access to popular desktop applications, applications specially suited for mobile users and basic communication services.

Wireless/mobile computing is a very challenging research area also due to the low data rates usually available. Moreover, wireless connections suffer a high variability in terms of bandwidth, are significantly less reliable than their wired counterparts, and, could be interrupted for various reasons (e.g., handovers). Additionally, the cost

[1] The emergence of nomadic computing was also facilitated by the rapid proliferation of portable computer equipment.

for a wireless data connection is increased: the cost per byte transmitted over the wireless interface is considerably higher than in wired infrastructures. Consequently, our focus should be centered on the reduction of data volume to be exchanged over the wireless medium. The efforts should concentrate on the development of a model able to support multiple types of applications, including current and emerging TCP/IP applications and terminal emulation.[2]

During the past years a number of efforts were made to consolidate the WWW with wireless communications. Notable examples are the MobiScape Project [2] and IBM WebExpress [3][3]. Our work is mostly based on the ideas integrated in the latter solution[4].

In this paper we discuss the design, implementation and performance evaluation of a mobile computing system that optimizes access to ***Web Services*** (WS)[5] and yields substantial benefits for the end-user. To the best of our knowledge there exists no equivalent client/intercept architecture for wireless WS.

The rest of the paper is structured as follows: In Section 2, we give a brief outline of WebExpress. In Section 3 we present the proposed architecture and discuss why WSs are conducive for the integration with an intercepting architecture and how they can be incorporated in a WebExpress-like system. We emphasize on the adopted object-oriented design along with the different roles assigned to the various classes/components, the functionality of the system, the protocol reduction, and, finally, the software components used for implementation. In Section 4, we elaborate on our experimental set-up, the metrics monitored and the collected statistics, and, finally, the same Section concludes this paper.

2 Prior Work

The use of WS in the wireless domain has been discussed extensively in [6]. The architecture scenarios discussed in the referenced paper refer to the operation of the mobile device either as requestor or as service provider.[6]

[2] The intercept model offers this advantage as it is transparent to both the client and the server, and therefore, can be employed with any client adaptation and be insensitive to the development of a particular client/server or communications technology.

[3] A more extensive survey of such systems can be found in [4].

[4] Both systems capitalize on the WWW proxy interface and use a similar architecture (also termed “intercepting technology”). WebExpress proposes an architecture that addresses the requirements of transaction processing through the adoption of “differencing techniques”.

[5] WS are a new breed of web applications following the Service Oriented Architecture (SOA) framework whose goal is to achieve loose coupling among interactive software agents [5]. A WS is a SOA with interfaces based on Internet protocols (e.g., HTTP), and messages encoded in XML (except for binary data attachment).

[6] In general, the protocols used in the wired WS world are inappropriate for the wireless medium. Therefore the wireless Internet community is pursuing their optimization. Commercial players like Microsoft and Sun introduce WS-specific features in their development frameworks and operating systems. In the recent past, optimized versions of SOAP and XML have been proposed for the wireless domain.

The basic architectural model of WebExpress consists of the Client Side Intercept (CSI) running in the end-user mobile device and the Server Side Intercept (SSI) running in the fixed network.[7] WebExpress's differencing optimization technique is applied on responses containing dynamic content (e.g., CGI output). Dynamic responses are not stored in cache but maintained as base objects: A common base object, associated with the resource, is created in both the CSI and SSI. Subsequent references to the resource will trigger, at the SSI, the computation of differences between its present form and the common base object. Such differences are transmitted over the radio interface. The CSI reconstitutes the referenced object and delivers it to the browser.[8]

3 Proposed Architecture

Description: At the top of the WS message stack is a mechanism for envelope extensions called SOAP headers.[9]

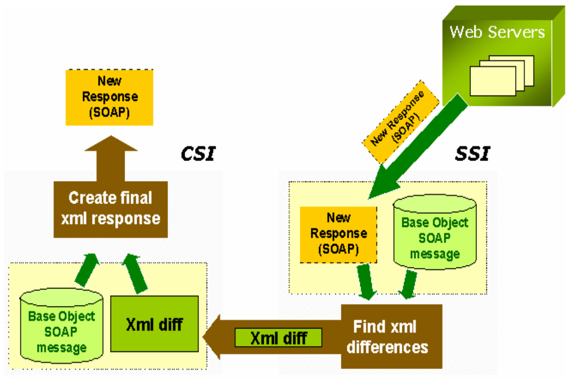


Fig. 1. Client/Intercept Architecture for WS

WS are the evolution of component-based systems such as DCOM, CORBA, RMI and EJB, that, also support distributed application logic. WS technology is considered more advantageous to these systems because it uses HTTP to be firewall friendly and payload agnostic, employs the more widely adopted XML scheme, uses the pervasive

[7] The CSI acts as a local proxy agent. The SSI is responsible for reconstituting a URL compatible data request and forwarding it to the designated origin server. On the reverse direction, the CSI reconstitutes the WWW stream and delivers it to the browser. Both the CSI and SSI incorporate caching mechanisms.

[8] HTML streams representing query responses usually contain a lot of unchanging formatting information (e.g., graphics, headers and footers). Responses from the same script are usually differentiated only by alphanumeric information.

[9] With SOAP headers, orthogonal extensions such as digital signatures can be associated with the body of the message contained in the SOAP envelope.

Internet concept of URLs to address object identification and offers more than a promise for interoperability as it exploits the openness of specific Internet technologies to address many of the interoperability issues of the previous systems.

Due to the above reasons and especially due to the HTTP/SOAP binding and use of URLs for identification, an optimizing mechanism for access to RPC-oriented[10] WS can be easily implemented with a Client/Intercept based system. In our proposal, there is an obvious analogy to the Client/Intercept model applied for the WWW. Instead of simple HTTP, now a SOAP call is packaged as the body of an HTTP request. In that sense, the differentiating engine is applied on the exchanged message, which is XML-encoded, and base objects are stored SOAP messages that essentially comprise a response for a WS, as depicted in Fig.1. In relation to the work in [6], our scheme can handle both a mobile device hosted WS client and server.

Object-Oriented Design: In this section we discuss also the object-oriented design of our implementation along with the functionality of constituent components.[11] **CSI Objects** *Client:* (CSI) Establishes the connection with the SSI and the browser, and initializes all other objects used. For each request received from the browser, it creates a thread (a ClientThread) to serve that request. *ClientThread:* Used to serve a specific request, and terminated as soon as this request is fulfilled. Many instances of this object, each of them serving a different request, may exist simultaneously. *Demultiplexor:* Resides on the CSI side and has the responsibility of the demultiplexing of the packages that arrive from the SSI and are destined to a specific browser. **SSI objects** *Server:* (SSI) One instance of this object is initially created, and is waiting for incoming connections from the CSI. For each incoming request, a new ServerThread is created to serve that request. *ServerThread:* Operating as a thread, this object is used to serve an incoming request from the CSI. It is generated by the Server object, and all its instances operate on the original threads of the SSI, and not on copies. We should stress out that the same applies for the ClientThread objects, which operate on the original objects of the Client. **Common objects** *BaseObjEntry:* Represents an entry for a base object. These entries are stored in the BaseObjHTable and include information such as: the URL for the base object, the filename that holds the data (xml code) for that base object, the CRC value of the object, and information about the time of storage and last access of this Base Object. *BaseObjHTable:* It is a hash table of BaseObjEntries, being used to store information about a specific base object entry. The key for the hash function is the URL of the base object. *BaseObj:* Manipulates the file system of the differencing subsystem. The files stored in a specific folder, are the base objects. The BaseObj controls the access to these files and performs a folder cleanup whenever the total size of files is greater than 80% of the maximum space specified by the administrator. Files are removed, until the total size of the cache is less than 60% of the maximum disk allotment.

[10] WS should be RPC-oriented, so that they apply to the case in which WebExpress has to deal with responses containing dynamic content.

[11] Since there are many similarities in the functionality of the CSI and the SSI, some of the following objects are used by both components.

Protocol Reduction: Towards the reduce of redundant volume exchanged, the techniques adopted are: *Virtual sockets:* The system minimizes overhead caused by opening and terminating multiple TCP connections, with the establishment of a single TCP connection between CSI and SSI used for all HTTP communication (i.e., requests and responses). The TCP connection is persistent across different transactions and alleviates the deficiency of HTTP's operation on top of TCP, which has much more negative effects over a wireless interface. In order to support a multiplexing functionality, we implemented a *virtual sockets* mechanism[12]. On the fixed network side, the virtual socket is associated with a normal socket on the designated WS provider. Some of the established virtual sockets are used for conveying commands related to normal socket calls (e.g., open, close) as a form of in-band signaling. *HTTP header reduction:* The ! "! # list of the types of documents that a given browser can handle usually remains constant. Therefore, the CSI allows this header information to flow towards the SSI only in the first request after the establishment of their connection and does not transmit it continuously.[13]

Functionality: Initially[14], there is no entry for a specific WS neither in the CSI nor in the SSI. When the first request comes, the CSI forwards it to the SSI, and the latter sends it to the WS provider. When SSI receives the response, it stores[15] it (and calculates the Cyclic Redundancy Check or CRC of it, which serves as a unique identifier for the message), before forwarding it to the CSI.[16] In the same way, it is stored at the CSI before being sent to the browser. At this point, a base object has been stored for the URL of the requested WS. Each base object is identified by the base URL. Let us

[12] In this mechanism, a small header that contains a virtual socket identifier prefixes data sent for a specific request. At the CSI, the virtual socket identifier is bound to a real socket at the browser. An actual TCP connection is established concurrently with the "opening" of the first virtual socket. It closes after a specific period elapses since the last virtual socket was closed. CSI and SSI are responsible for the role assignment according to the virtual socket. CSI receives requests from the browser and forwards them to the SSI. For every received request, the SSI establishes a connection with the respective WS provider and forwards the request. As soon as it receives the response, it closes the connection with the server, sends the document to the CSI via the TCP connection, but does not terminate it. CSI forwards the document to the browser, and closes the connection with the latter.

[13] Both CSI and SSI maintain this Accept List as part of the connection status information. For every request received from the browser, the CSI checks the list, to verify if it is identical with the stored one, and if so, the list is omitted from the request, but subsequently inserted by the SSI. In case there is another list in the request, SSI stores the new list.

[14] It should be mentioned here, that, in the first place, when the CSI receives an HTTP request, it checks if it is a request for a WS. The check is conducted in the following way: if the binding method between SOAP and HTTP is GET, the suffix of the URL is checked. If, on the other hand, the used method is POST, and consequently, there is a body in the transmitted message, the CSI soap proxy checks, with the help of JDOM XML parser, if the body of the received SOAP message (an XML document) is well formed. If this check is successful, it forwards the message to the SSI and the processing continues. Otherwise, an HTTP error message is created and the request is not processed any further.

[15] The storage takes place on scope of the entire SOAP message transmitted.

[16] In the framework of the synchronization schema implemented between the CSI and the SSI.

suppose that there comes another request for the same WS. As soon as the request is intercepted, the CSI proxy checks if it has its URL stored. If a version is found, CSI forwards the request along with a relevant indication of existence and the CRC value of that base object. In every case, a message is sent to the SSI and the SSI always communicates with the WS providers. If there is no base object with the specific CRC at the SSI, SSI notifies CSI that it has to store it. Otherwise, if the CRC of the CSI base object is identical to its counterpart at the SSI, the differencing engine is engaged. If the files are not identical (i.e., different CRC values), SSI replaces the existing base object with the new one and notifies CSI to do the same.[17]

Implementation Details: As aforementioned, our system was implemented in Java. We used the Apache Ant and Log4j tools for building management and logging mechanisms respectively, as well as the JDOM parser for check of XML request syntactical correctness. The graphical user interface (implemented with Java Swing) allows the administrator to define operational parameters, and view the accumulated statistics along with information concerning base objects. In order to implement the differencing mechanism, we employed Microsoft XML Diff and Patch, a set of tools for comparison of two XML documents and application of the changes (patching).[18]

4 Performance Assessment

The performance of the system was assessed through a series of experiments (1000 WS requests performed per experiment).[19] The 40%, 60%, 80% and 100% respectively of full storage capacity of our proxies was exploited. In every case, the LRU algorithm was enforced as soon as the capacity of stored objects became greater than 80% of the current proxy capacity. Additionally, two series of experiments were implemented: one with 10 distinct WS (randomly chosen by the Client) and one with 20 WS. The monitored statistics included, among others, the Local/Total Bytes Read Ratio, that practically indicates the efficiency of the system by denoting the ratio of bytes that the CSI delivered to the browser against bytes retrieved from the SSI by the CSI. The accumulated statistics are presented in Table 1 and Table 2.

[17] For even more optimization, the differencing stream (i.e., the difference between the base object stored at the SSI and the answer from the server) is sent to the CSI-instead of the response from the origin server-only on the presumption that the size of the resulting stream of the differencing engine is less than a percentage (70% in our implementation) of the size of the actual xml document, which would be sent. Otherwise, the retrieved response is directly forwarded to the CSI. The differencing stream (which is also XML-encoded) is sent to the CSI. At the CSI, an entity update engine uses the differencing stream along with the stored base object of the request, in order to patch the new response (at the CSI rebasing takes place), and ultimately forward it to the browser.

[18] XML Diff is based on the Document Object Model (DOM) and detects addition, deletion and structural changes like the removal of an XML sub-tree. It produces *Xml Diff Language Diffgram*, an xml-based concrete and short output language, which can then be used to perform a patch operation using the XML Patch.

[19] There was no need to perform a large number of requests to "warm up" any cache, so the above-discussed number of trials has been considered adequate.

Table 1. Experiment results for 10 available WS

Ex-periment No.	Base Object Size (MB)	Percentage of Proxy Capacity Used	Base Object hits	Total Bytes Read (CSI)	Total Bytes Read (SSI)	Bytes Read Ratio %
1.	25131	40%	231	4906932	6179314	79,409
2.	37696	60%	335	4387517	6477379	67,736
3.	50261	80%	448	3821632	6246173	61,184
4.	62826	100%	580	3356139	6567688	51,101

Table 2. Experiment results for 20 available WS

Experiment No.	Base Object Size (MB)	Percentage of Proxy Capacity Used	Base Object hits	Total Bytes Read (CSI)	Total Bytes Read (SSI)	Bytes Read Ratio %
1.	39070	40%	239	4007997	4816465	83,215
2.	58605	60%	355	3651059	4991831	73,141
3.	78140	80%	455	3136082	5172683	60,628
4.	97674	100%	588	2538281	5255106	48,301

It can be observed that the traffic reduction was substantial (approximately 40%), an effect that was anticipated: the number of bytes read at the CSI was more than 40% of the bytes read from the SSI. As the number of used proxy capacity increases, so does the Bytes Read Ratio[20] too, since the possibility of hitting an existing base object increases, and thus there is no need for data exchange.

Having realised a series of experiments with the purpose of evaluating the alleged benefits from the various optimisation techniques, a series of experiments that demonstrated a considerable improvement in the use of WS resources (with the indicative example of 40% gain in the volume of information flowing over the wireless interface between the CSI and the SSI), we conclude that our WS-oriented client/intercept system contributes to decreased latency, decreased bandwidth use and other efficiency benefits.

References

1. T.F.La Porta, et al., “Challenges for Nomadic Computing: Mobility Management and Wireless Communications”, ACM Journal of Nomadic Computing, Vol.1 No.1, 1996.
2. C.Baquero et al., “MobiScape: WWW Browsing under Disconnected and Semi-Connected Operation”, proceedings of the 1st Portuguese WWW National Conference, Braga, Portugal, July 1995.

[20] The ratio of bytes read at the CSI to bytes read from the SSI. It should be noted, however, that this percentage largely depends on the differencing mechanism incorporated in the system.

3. B.C.Housel, and D.B.Lindquist, “WebExpress: A system for Optimizing Web Browsing in a Wireless Environment”, proceedings of ACM/IEEE MobiCom ’96, NY, USA, Oct. 1996.
4. S.Hadjiefthymiades, and L.Merakos, “A Survey of Web Architectures for Wireless Communication Environments”, Journal of Universal Computer Science, Vol.5, No.7, Springer Verlag, 1999.
5. Thomas Erl, "Service-oriented Architecture: A Field Guide to Integrating XML and Web Services", Prentice Hall, April 2004.
6. T.Pilioura, A.Tsalgatidou, S.Hadjiefthymiades, “Scenarios of using Web Services in M-Commerce”, ACM SIGecom Exchanges, Vol. 3, No. 4, January 2003.

Author Index