Modeling Component Erroneous Behavior and Error Propagation for Dependability Analysis

Naif A. Mokhayesh Alzahrani and Dorina C. Petriu

Department of Systems and Computer Engineering, Carleton University
Ottawa, Ontario, Canada K1S 5B6
{nzahrani,petriu}@sce.carleton.ca

Abstract. Modeling erroneous behavior of software components along with normal behavior tends to be complex and hard to read or modify. However, ignoring the erroneous behavior and error propagation in models used for dependability analysis has a negative impact on the dependability assessment accuracy. In this paper, we propose a framework for automating dependability modeling and analysis that considers component erroneous behavior. Particularly, the paper focuses on our Component Erroneous Behavior Aspect Modeling approach (CeBAM), which captures component erroneous behavior and error propagation. We apply aspect-oriented modeling techniques to model erroneous behaviors separately from the normal behavior. The approach reduces the model complexity and improves its readability and modifiability. In addition, we propose a profile to extend the UML protocol state machine to capture both incoming and outgoing messages on components' ports. We automate the composition of normal and erroneous behavior by aspect weaving. This enables the next step: conformance verification between each component's complete internal behavior and its protocol state machines, as well as between component interfaces.

Keywords: erroneous behavior model, error propagation, aspect-oriented modeling, conformance verification.

1 Introduction

Model Driven Development (MDD) is a promising approach for software development that changes the focus from code to models. This change of focus facilitates also the analysis of different Non-Functional Properties (NFP) using formal analysis models obtained by model transformations from the software models. For instance, in this paper we are interested in the analysis of dependability attributes (such as reliability and availability) using analysis models automatically generated from software models extended with dependability annotations. In [1] the authors survey the works on dependability modeling and analysis based on UML and UML extensions for annotations. Another software development paradigm of interest is Component Based Development (CBD), which applies the "divide and conquer" principle to manage system complexity. Each component is a unit of composition that interacts with other components through

F. Khendek et al. (Eds.): SDL 2013, LNCS 7916, pp. 124–143, 2013.
© Springer-Verlag Berlin Heidelberg 2013

predefined interfaces. CBD is a reuse-based approach that has an impact on the development time and system dependability attributes.

Combining MDD and CBD is an appealing approach for the development of real-time embedded systems, as it reduces the complexity, time and cost. In addition, MDD and CBD help to integrate dependability modeling and analysis during the design phase. The quantitative results of these analyses will support the developer in taking the right decisions for building dependable systems. Different approaches were proposed in literature to address reliability and availability modeling [1,2]. However, many existing approaches do not adequately consider error propagation in predicting system reliability [3].

The long-term goal of our research is to propose a framework based on standard modeling languages (such as UML and QVT), which would help developers to evaluate dependability attributes during a CBD + MDD process, taking into consideration component erroneous behavior and error propagation. We believe that including component erroneous behavior in dependability analysis and prediction will help developers to take the right design decisions. For instance, selecting proper fault tolerance mechanisms, placing error detection and using suitable recovery approaches are examples of critical decisions taken in the design phase based on quantitative values. The findings of [4,5] support this belief, since they show that error propagation may have significant impact on reliability prediction. Thus, in our approach the evaluation of dependability attributes is based on component behavior (normal and erroneous).

A software component has two views: internal and external. An internal view represents component's private properties realizing the provided services. An external view shows the public properties of the component in terms of provided and required interfaces. Modeling erroneous behavior of these views along with normal behavior in one model tends to be complex and hard to read or modify. Moreover, it is not easy to capture error propagation between components using existing behavior models such as UML2 state machines. As a result, developers often focus on the normal behavior of both views and tend to ignore the erroneous behavior.

To overcome these difficulties of modeling erroneous behavior and error propagation in CBD, we introduce the Component Erroneous Behavior Aspect Modeling approach (CeBAM). It captures the component erroneous behavior separately from the normal behavior. CeBAM uses aspect-oriented modeling [6] to simplify modeling the erroneous behavior and to automate its composition with the normal behavior.

CeBAM uses UML state machines to represent the component internal normal behavior and extended protocol state machine for port and interface behavior. Normally, UML protocol state machines capture only incoming messages, so we defined a profile to capture both incoming and outgoing messages. Another UML extension developed in this paper is an erroneous behavior profile to capture the chain of dependability threats for the component internal behavior, as well as for its ports and interfaces. In addition, this profile shows the error propagation from the component internal behavior to its ports and further to other components.

One of the CeBAM advantages is that it provides an easy and practical way to model component erroneous behavior separately from the normal behavior, using aspect-oriented modeling [6]. Indeed, this reduces the state machine complexity and makes the model easy to read and maintain. In addition, the automated composition of erroneous with normal behavior will be further used for: a) conformance verification between the internal behavior of each component and its protocol state machines, and between components interfaces, and b) derivation of dependability analysis model.

This paper is organized as follows. Section 2 presents our long-term objective to automate dependability modeling and analysis. In section 3 we explain briefly the Emergency Monitoring System (EMS) case study used in this paper. The CeBAM approach is introduced in section 4. Related work is discussed in section 5. In the last section, we conclude and summarize our ongoing work.

2 Overview of Dependability Analysis Framework

The long-term objective of our research is to provide software developers with automated techniques for architectural-based software dependability modeling and prediction. This paper is the first step on the road toward such an objective. Figure 1 illustrates the overall activities of our proposed framework in order to provide context for the contribution of this paper and to put it into perspective. We start with a component-based software architecture model that needs to be evaluated in terms of reliability and availability. For the most important scenarios, we identify the involved components and the interaction between them. Next, we build component behavioral models using CeBAM that considers erroneous behavior of the involved components. This model is also enriched with dependability annotations using the DAM profile [7]. Note that we do not show the dependability annotations in our case study because of the limited space. Aspect oriented modeling (AOM) approach [6] is adopted to allow for more flexibility in modeling erroneous behavior and to provide an automated composition of the erroneous with the normal behavior.

Reasoning on behavioral compliance of a component-based software architecture is required to validate the software architecture [8]. Thus, in our approach, after composing the normal and erroneous behavioral models, we validate the conformance in two stages. The first is conformance validation between component internal behavior and its protocol state machines. A mismatch would impact negatively the component reliability, since the internal behavior would receive unexpected messages that cannot be handled. So, any detected mismatch must be corrected. Second, we verify the compatibility between the components' provided and required interfaces.

In the literature, different approaches are suggested to check for component conformance by finding deadlocks in the formal model that is transformed from the main software model [9,10]. Different formalisms may be used, either various kinds of formal logic or of Petri nets. Since we are planning to use state-based dependability analysis based on Petri Nets, similar to the approach in [11], we

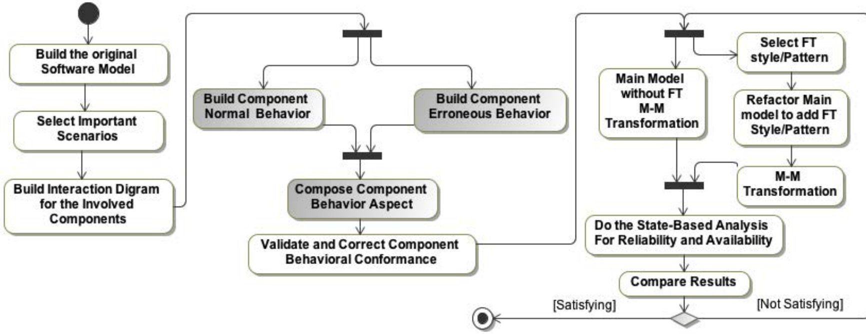


Fig. 1. Overview of the proposed framework (Shaded activities are the focus of this paper)

will also use a class of Petri Nets, namely Stochastic Reward Net (SRN) [12] for conformance validation instead of a model checker. We chose SRN because its marking dependency property helps in obtaining more compact models for complex systems, which helps in limiting the size of the state space during the analysis. Currently, we are working to automate the transformation of component behavioral model to SRN in order to validate conformance and predict reliability and availability as well.

Adding a fault tolerance mechanism may improve the system reliability, but the effects are non-trivial and depend on the context [13]. As shown in Fig. 1, our approach aims to provide developers with automated tools to assess the reliability (availability) of different fault tolerance mechanisms applied to the system under evaluation. By comparing the predicted results, a developer can select the best design alternative.

Automation is one of the key features in our approach. We utilize Query View Transformation (QVT) [14] to automate the composition of component erroneous behavior with the normal behavior (based on aspects) and to generate SRN models for conformance checking. Also, the process of refactoring the main architectural model by adding a selected fault tolerance mechanism from a predefined collection of fault tolerance styles (see Fig. 1) can be automated with the help of QVT model transformations. Moreover, automated model transformation will generate the SRN model used for state-based dependability analysis for the architecture without and with fault tolerance mechanisms. Comparing the results, the developer will be able to evaluate the effects of the selected fault tolerance style on the overall system reliability and availability.

3 Case Study

Emergency Monitoring System (EMS) is the case study that will be used throughout this paper. This case study was developed in [15] and our goal is to improve the design by modeling the normal and erroneous behavior of each

component. The erroneous behavior will capture any locally activated fault and show how it propagates to the connected components through its ports.

The EMS is a distributed system. It consists of a central monitoring service, operator presentation service and several distributed monitoring sensors. In addition, in some remote areas, a remote monitoring system is installed with its sensors. All the remote systems and distributed sensors are reporting regularly the current status of the external environment to the central monitoring service. The monitoring service stores and updates the recent status and presents it to the operator. As a result, the operator will have the updated status of each site and, accordingly, he can take action if the emergency alarm is detected.

COMET methodology [15] was used in this case study. Figure 2(a) shows the use case model and Fig. 2(b) the distributed component-based software architecture of EMS. Due to limited space, we select only one scenario called *generate alarm* to illustrate our approach for modeling the component behavior. In this scenario, three components are involved. Figure 2(c) shows the interaction between these components in case of generating an alarm. In general, component interaction takes place either via method calls (synchronous) of provided and required interfaces or via notification signals (asynchronous). According to [16], if an internal fault is activated but not properly handled inside the component, then this fault will end up in a failure, which will propagate to other components that depend on it. For instance, in the selected scenario, if the *AlarmService* component has failed due to an internal exception or hardware failure, the manifested failure will be propagated to the monitoring sensor (see Fig. 2(d)). Moreover, the detected emergency alarms cannot be reported to the operator since the core component *AlarmService* is down.

4 Component Erroneous Behavioral Aspect Modeling (CeBAM) Approach

A software component has two views: internal and external. The internal view represents component's private properties realizing the provided services. The normal behavior of this view can be described using UML behavioral state machine (BSM) [17]. An external view shows the public properties of the component in terms of provided and required interfaces. Interactions between components are actually methods calls (synchronous) or exchanging notification messages (asynchronous). Protocol state machine (PSM) can be attached to each interface to describe the legal sequence of operations calls [17].

A fault may be activated inside a component and then propagated to the interfaces and then to all dependent components if it is not handled internally or at the component port. Moreover, each fault type may have a different propagation path. In fact, modeling both the internal and external view of the component erroneous behavior will help to improve the software design. Unfortunately, BSM and PSM do not allow for an easy and practical way of modeling normal and erroneous behavior, due to model complexity and a lack of ability to capture error propagation.

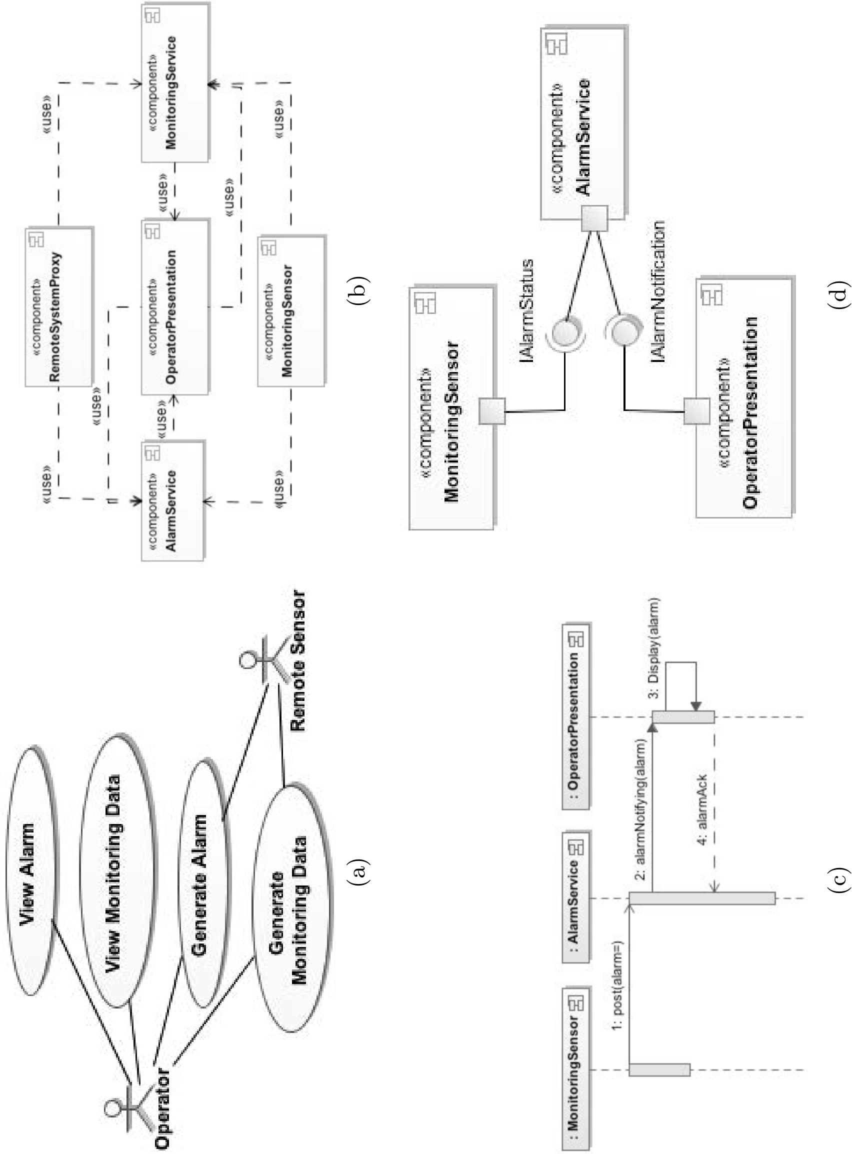


Fig. 2. EMS case study [15], (a) Use cases, (b) Component for the complete system, (c) Interaction of the generate alarm scenario, (d) Components involved in generate alarm scenario

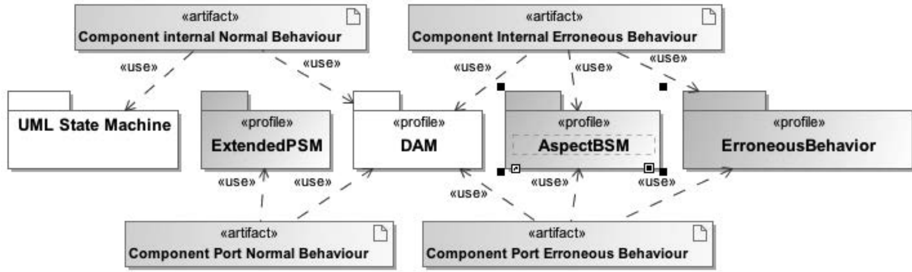


Fig. 3. Profiles and artifacts in CeBAM

In CeBAM, component behavior consists of normal and erroneous behavior that describe both component interfaces, ports and internal behavior. In [1,2] different approaches are presented for dependability modeling and analysis, and we noticed that most of these approaches focus only on the normal behavior, ignoring the erroneous behavior due to its complexity.

Our objective in this paper is to provide a practical solution to model complete component behavior for the internal and external views, by considering erroneous behavior and error propagation. The CeBAM approach is developed to realize this objective. Figure 3 shows the new profiles and artifacts used in CeBAM. For all defined profiles we followed the approach from [18], which suggests to start with defining the domain model as a starting point and then mapping the domain model concepts to the UML2.x meta-model, in order to identify the new stereotypes and attributes.

In CeBAM the internal component behavior is modeled using BSM according to the provided and required services. For instance, Fig. 4 shows the internal normal behavior for a single service of *AlarmService* component in the case study. Normal behavior of component ports or interfaces is modeled using extended PSM (as described in section 4.1).

The ErroneousBehavior profile and AspectBSM are used together to model the erroneous behavior (both internal and external views) separately from the normal behavior as aspect models. Then we automate the composition of the erroneous behavior with the normal behavior of both views (sections 4.2, 4.3 and 4.4). Since we consider also protocol state machines, we validate the conformance between component behavior and its PSMs, as well as between components' interfaces (section 4.6).

In CeBAM we adopt the aspect oriented modeling approach [6] to model component erroneous aspect. Actually, we consider erroneous behavior as a crosscutting concern that can be modeled separately and then we automate the composition with the base model (i.e., the normal behavior for both views). Using this approach, a developer will not have to learn new concepts in order to model the erroneous behavior with the dependability annotations. Additionally, there is full flexibility to update or change any behavior separately, since the composition of the complete behavior is automated.

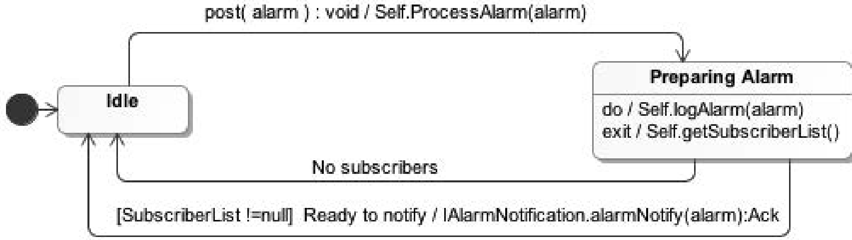


Fig. 4. Partial BSM of internal behavior of *AlarmeSerivce* Component

4.1 Extending Protocol State Machine

According to [17] a PSM is a specialized behavioral state machine defined in the context of a classifier, that can be used to specify which operations of the classifier can be called in which state and under which conditions. PSM is used to describe only the legal usage of any classifier. PSM does not show any specific behavioral implementation since actions are not allowed on transitions or in the states. Actually, states in PSM do not have entry, exit, do activities. On the other hand, composite state and concurrent regions are allowed, but history pseudosates are not. (We are not using composite states or concurrent regions in this paper).

A protocol transition captures the legal transition of the context classifier. It has a pre-condition, a trigger, and a post-condition. Protocol transition shows that the associated operation can be called under a specific condition (precondition) and then after the complete execution the destination state can be reached if the post condition is satisfied. Moreover, PSM inherits run-to-completion semantics from BSM, i.e. the action on a transition is uninterruptible. This implies that no other event can be accepted during the transition. For instance, if a fault is activated during the execution of the called operation, the transition will not be completed. Additionally, nested calls cannot be captured.

Due to the restrictions applied to PSM, only unidirectional communications can be captured [8,17]. For instance, in Fig. 2(d) we can use PSM to model the communication of the provided interface connected to the *AlarmService* component. In this case PSM can only capture incoming calls to that interface. Moreover, for each interface we have to create a separate PSM since it can only capture the communication on a single classifier.

In UML, a port is a property of a classifier [17]. A port can be associated with a component (i.e., the UML classifier) to specify an interaction point between the components and its environment and between component and its internal parts. A combination of required and provided interfaces can be associated with a port; thus, a port may specify provided services to other components as well as required services. In the case study, we assume that each component has only one port for interaction with the environment; thus, the external view of the component is actually the port behavior.

We can describe the external view of the component behavior by describing its protocol state machine. As mentioned before, PSM can be used to precisely capture that behavior, but it captures only a single direction (incoming) and it does not allow recursive calls due to the run-to-completion semantics. To overcome these limitations, we extend the PSM by introducing a new profile as shown in Fig. 5. This profile will be used to model "extended" protocol state machines. *PortTransition* stereotype is extending the *ProtocolTransition* metaclass with different attributes.

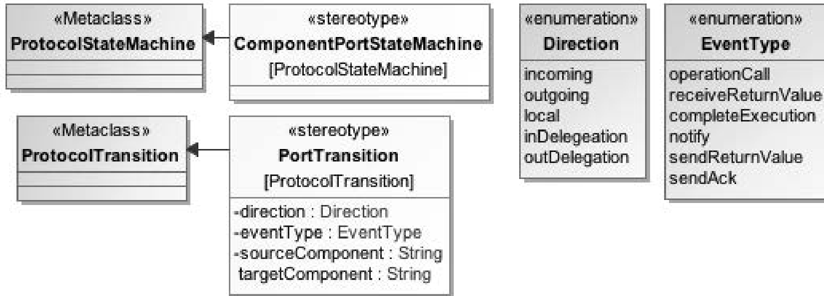


Fig. 5. ExtendedPSM profile

In *PortTransition* we can capture the direction of each passed message, either sent to an associated component or received from the environment. Sometimes the PSM state changes because of an internal event.

In this profile we respect the run-to-completion semantics and we can show atomic events. For each event in *PortTransition* we specify the direction (incoming, outgoing and delegation) and the type of that event (operation call, notify signals, receive return value from the called operation locally or externally and complete execution signals). Moreover, we show the source and target component associated with that event.

The *AlarmService* component of the EMS case study has two provided interfaces and one required interface. These interfaces are associated to a single port (see Fig. 2(d). To describe the external view of the component, we use UML and the ExtendedPSM profile to model the protocol state machine (see Fig. 6). Initially, it receives incoming message from the monitoring sensor to execute one of its provided services *post(alarm)*. The *post* method is implemented by the *AlarmService* component and therefore, different actions will be done internally, for instance, storing the reported alarm and then notifying the operator. These actions will change the state of the PSM, as precisely captured using the ExtendedPSM profile. In Fig. 6, each transition of the alarm service PSM is atomic and it has run-to-completion semantics. In addition, the direction and the source of the messages are also captured.

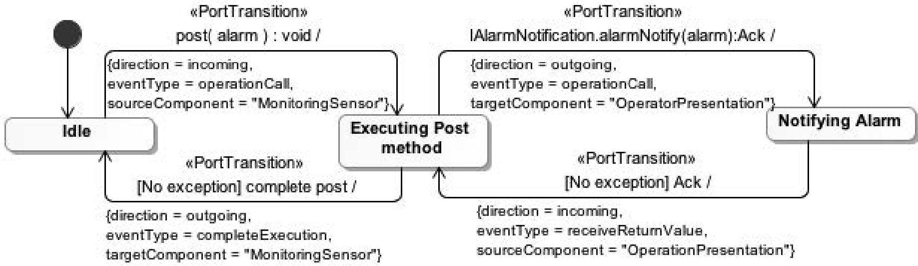


Fig. 6. Fragment of *AlarmService* normal behavior protocol state machine

4.2 Aspect Composition

We adopt AOM mechanism [6] to model separately component erroneous behavior and then to compose it with the normal behavior. Figure 7 shows the domain model for the proposed AspectBSM profile and Fig. 8 shows the actual profile according to the domain model.

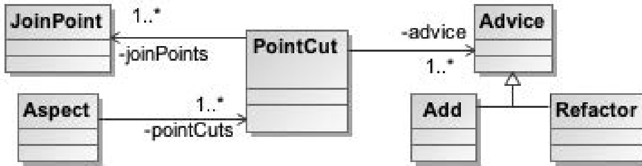


Fig. 7. Aspect domain model in erroneous behavior context

This profile is based on the main concepts of aspect-oriented modeling and is similar to the approach from [19]. *Aspect* describes a crosscutting concern; in our context, the aspect will be the erroneous behavior of both component views. For each crosscutting behavior we have a *pointCut*, which is a condition of a query that identifies the place(s) where the new behavior should be added in the base model or which model element needs to be refactored. A candidate element in the base model that corresponds to a *pointCut* is called *joinPoint*. In other word, the *pointCut* will select one or more *joinPoint* where the new behavior can be applied. In our approach, the *pointCut* will be an OCL query that selects states or transitions. Note that we will not add any new stereotype in the base model to identify the *joinPoint*. *Advice* is a new behavior introduced to the base model at the *joinPoint* and it could be *add* or *refactor advice*.

As mentioned before, transitions in behavioral state machines have run-to-completion semantics according to [17]. In some cases, the action associated with a transition is an operation call. The operation must be executed successfully before entering the new state. However, during its execution, faults maybe activated that will interrupt the transition. Our objective is to model any fault

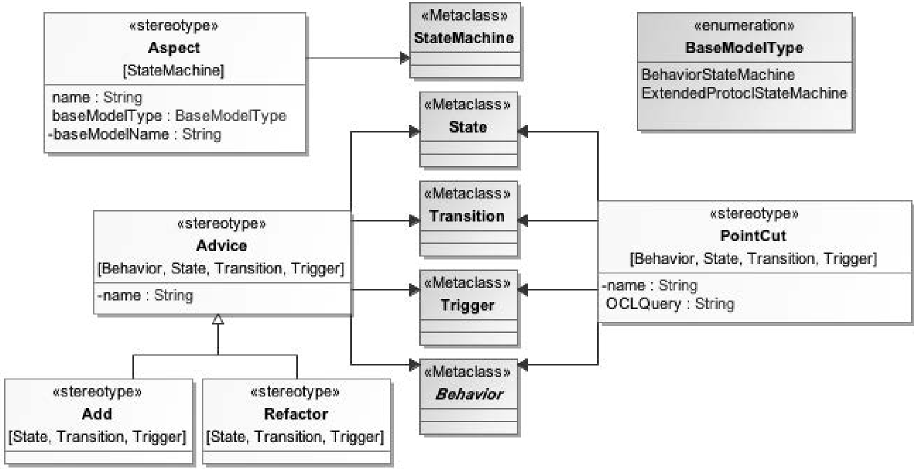


Fig. 8. AspectBSM profile

that may be activated during the transition, but at same time we want to respect the run-to-completion semantics. To achieve that, we introduce a *refactor* advice applied to the respective BSM transition. Before adding the erroneous aspect of that operation we should introduce a new state called *intermediate state* and a new transition called *done*. Figure 9 illustrates an example of the *refactor* aspects applied to the internal behavior of Alarm Service component.

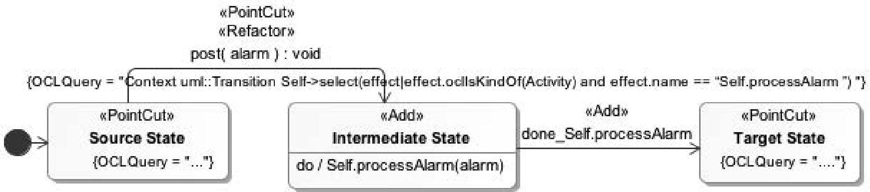


Fig. 9. Refactor aspect of *processAlarm* activity

For instance, *processAlarm(alarm)* is an operation executed as an effect of the transition from the state *Idle* to *PreparingAlarm* in the base model (see Fig. 4). First we identify the source and target states of that transition and then we add a new state (called *IntermediateState*) reached from the source state with the original transition, but without the call to the effect operation. We move the *processAlarm(alarm)* operation to the newly introduced state as a *do activity*. Finally, we add a new transition *done* from the *IntermediateState* to the target state. This new transition represents the successful execution of the transition action from the base model. In this way, we preserve the run-to-completion semantics and we can add later an erroneous transition from the *IntermediateState*.

We use the *refactor* aspect only in BSM describing the internal behavior of a component, but we do not need refactoring in the protocol state machines developed with ExtendedPSM because all transitions are already atomic.

4.3 Modeling Component Erroneous Behavior

Different error states and failure modes can be identified for a single component. Each failure may have a different propagation path. Our objective is to model the error propagation between components separately as crosscutting concerns in order to study how this propagation impacts the overall reliability and availability of the system. We develop a new profile to model the component internal and external erroneous behavior, as well as the error propagation between components. Figure 10 shows the profile stereotype and attributes. This profile captures the two kinds of states and transitions: error, failure mode states and erroneous, recovery transitions. For each transition type it depicts the direction, event, and source operation and target operation. Using this profile we can model different kinds of failure caused by software exceptions or hardware failures. In CeBAM we use this profile and AspectBSM profile together to model erroneous internal component behavior and PSM erroneous behavior as aspects, separately from the normal behavior. This approach will allow for flexibility in the modeling of erroneous behavior, creating models easy to read and modify.

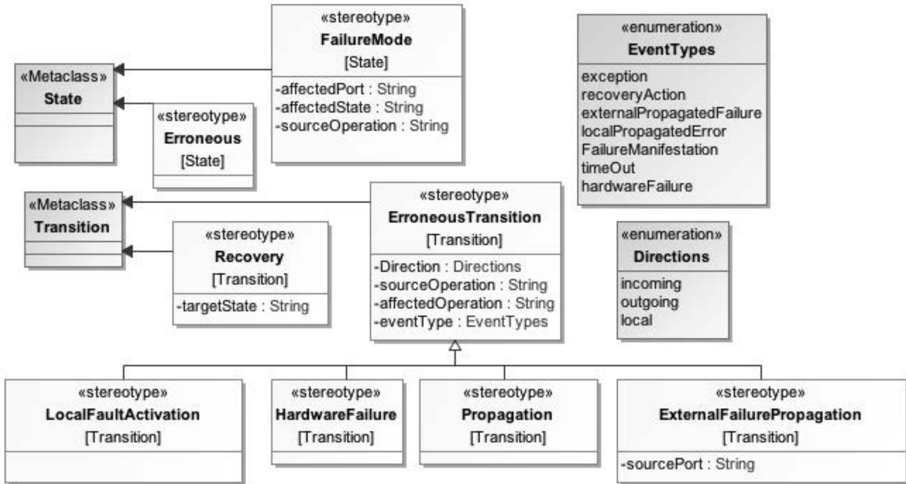
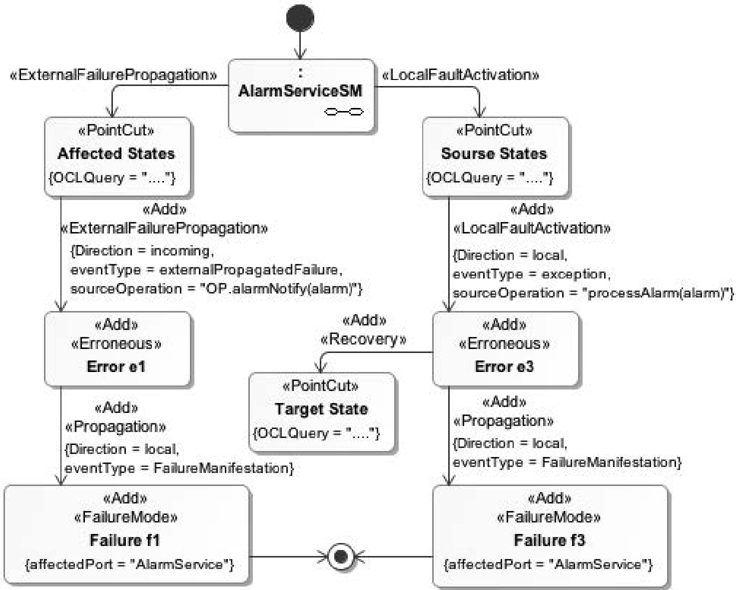
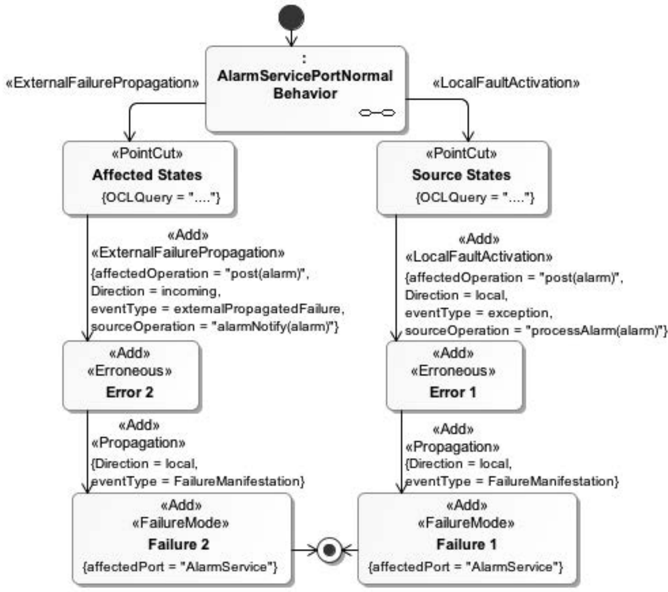


Fig. 10. Erroneous Behavior profile

For instance, Fig. 11 (a) shows different errors and failure modes activated internally in the *AlarmService* component. A local failure is activated because of an internal exception occurring in *processAlarm(alarm)* method and an external failure is propagated from *OperatorPresentation* component. The internal failure



(a)



(b)

Fig. 11. Fragments of *AlarmService* erroneous behavior: (a) internal (a) external

will be propagated to the component port and then to the connected component causing another error and failure types according to [16]. In Fig. 11 (b) we notice that the external failure is captured as well in the component port behavior and propagated to the internal component behavior. The profiles in CeBAM were designed to capture all required details described in [3,11,16] to model component internal and external erroneous behavior.

4.4 Behavior Composition

We follow AOM [6] approach to compose both behaviors. Composition directives are describing the sequence and the order in which aspects need to be composed with the base model. For instance, in behavioral state machine of component internal behavior *refactor* aspects will be processed and applied first before any *add* aspects. Figure 12 shows fragments from the BSM and PSM for *AlarmService* after composition. The states shaded in gray are erroneous states.

4.5 Guidelines for Using CeBAM

Modeling component behavior using CeBAM can be done in two phases. In the first phase we just model the normal behavior of both component views (internal and external). BSM will be used for the component internal normal behavior (see Fig. 4) and extended PSM for the external view (see Fig. 6). The second phase is focusing on modeling component erroneous behavior separately using two profiles: AspectBSM and ErroneousBehavior profile. The outcome of this phase is represented by two aspect models: one for the erroneous behavior of the internal view and another for the external view. We may need a few iterations to build these two models. First we capture the local failures and then in the next iteration(s) we may have to add propagated failures that originated in other components. The iterations will end when all errors/failures have been "propagated". In some cases we may need to create refactor aspects to preserve the run-to-completion semantics of BSM transitions.

4.6 Components Behavior Conformance

A system is built from different component interacting with each other to accomplish specific scenarios. In UML 2 [17] conformance is considered, but the definition of the conformance semantics is limited [8]. Several approaches in the literature propose different solutions for checking conformance. For instance, in [10] labeled Petri net are used to check the compatibility between component interfaces. In [9] a model checker is used to automate conformance validation between components PSM and its internal behavior in the context of UML-RT.

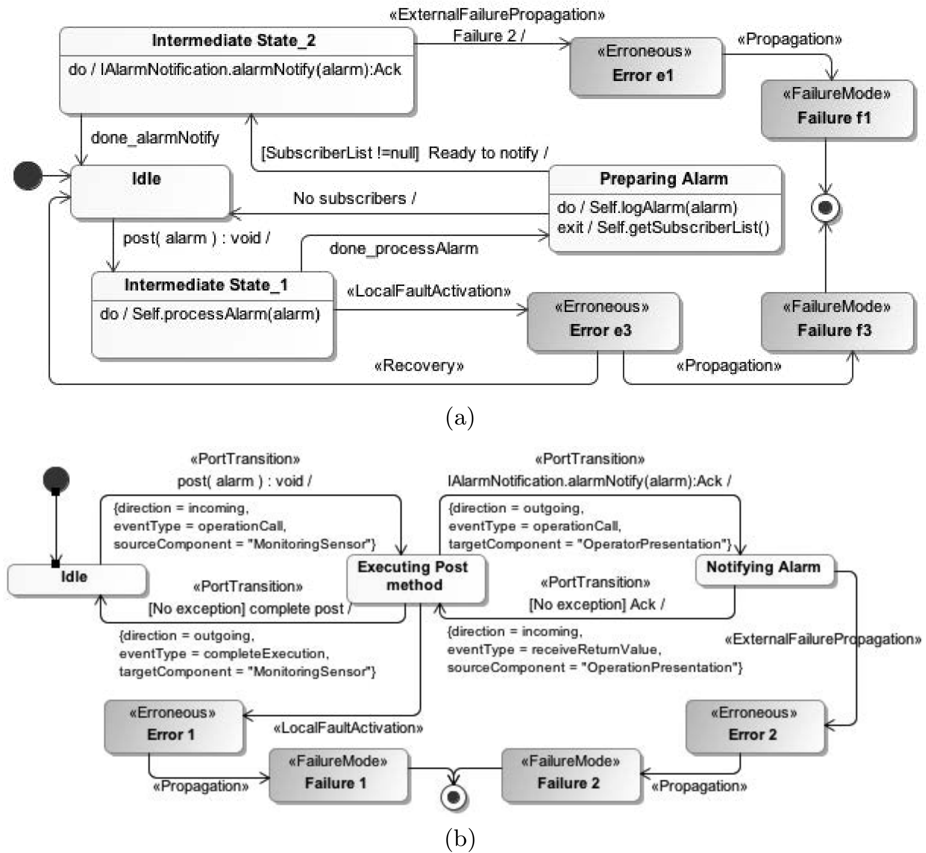


Fig. 12. Fragments of *AlarmService* complete component behavior after composition: (a) internal behavior, (b) external behavior

In our case we are working to address the conformance validation in two levels: between required and provided interfaces and between protocol state machines and internal component behavior. This conformance validation will be done for the composed state machines, which capture both the normal and erroneous behavior. In the first stage, we will automate the conformance validation between internal component behavior and its ports to fix any incompatible behavior. In the next stage we will check the conformance between components interfaces.

5 Related Work and Discussion

Different approaches can be found in literature for predicting the reliability and availability of component based systems. The proposed approaches are classified in [20] into states-based, path-based and additive models. We are following the

state-based approach in our work. Our approach considers that erroneous behavior is an important part of the dependability assessment. It was inspired by different works, such as [4,5,11,19,21]. In [19] a methodology for modeling system robustness behavior using aspect-oriented modeling is proposed. The authors defined an aspect profile for robustness behavior which inspired our definition of the AspectBSM profile. However, we customized our profile for modeling component erroneous behavior.

Some limitation of the UML behavioral and protocol state machines for capturing component port behavior are identified in [8]. The author proposes a Port State Machine (PoSM) to capture the interleaving operation calls on a port, which is defined by modifying the UML meta-model, specializing some of the meta-classes. PoSM focuses only on the operation calls between components and does not capture other type of triggers, i.e. failure propagation and signals. PoSM is not supported by current UML2 tools since the UML meta-model was changed, and this is a major limitation for its applicability. Our approach is addressing the same basic limitations of PSM, but we utilize the UML2 profile mechanism for the required extensions, to make our approach supported by existing UML tools.

A new development framework for dependability analysis was introduced in [11] based on a new intermediate dependability-specific modeling language CHESS ML. A component error model view, represented with a special kind of state machine in CHESS ML, shows faults that can be activated internally or propagated from other components. CHESS ML models are transformed into Petri nets for state-based reliability analysis. Many differences can be identified between our approach and the CHESS approach. First, in our case we do not use an intermediate model and we plan to transform to Petri net directly from the base model. Second, our approach considers the origin of the fault in the normal behavior and uses composed state machines for conformance verification and for generating the Petri nets models. Last but not least, we are following the UML standard for our software models.

The importance of including error propagation in the reliability assessment was identified in [4,5]. The work in [4] considers an error propagation probability and proves the significant impact of error propagation in reliability predictions. The approach in [5] takes into account the error propagation and error propagation path, but does not consider fault tolerance. In [3] a framework for compositional reasoning on the error model is proposed; a new error classification and failure modes are introduced. The execution environment and component usage profile are considered in [22]. The work in [13] studies the effects of software fault tolerance mechanisms in varying architectural configurations in models built in the Palladio Component Model (a non-standard software modeling language for component-based systems). A framework for predicting component reliability by studying component internal behavior is developed in [23], but it focuses on internal components and ignores the error propagation.

The approach in [24] employs application blocks to represent application functionalities and the internal behavior is described using activity diagrams.

An application block may include an activity and be a part of another one. External State Machine (ESM) and its extended version (EESM) are used to model the UML pins behavior, which model the interaction between activities. In our CeBAM approach we follow a component-based approach, using standard UML components with ports and connectors to represent the system architecture. The component internal behavior is modeled using state machines and we extend the UML protocol state machine to describe the component port behavior according to the required and provided interfaces for both incoming and outgoing messages. Moreover, in [24] External Reliability Contract (ERC) are introduced to describe the failures during the communication between activities. The ERCs are modeled separately and they are composed with EESMs using an aspect-oriented modeling approach to reduce the modeling complexity. In CeBAM we also use AOM for the same reasons. However, in CeBAM the erroneous behavior will capture the source of failure inside a component and it shows how it propagates and affects other connected components. In [24] a model checker is used for formal verification, while in our case we are working on transforming the UML model to Stochastic Reward Networks (SRN) [12] for both dependability analysis and conformance checking.

The separation of concerns principle is applied in component-based architectures to reduce the complexity and to improve the quality. This principle is realized by separate protocol behaviors that describe the provided and required component functionality. However, the challenge consists in the composition of these separate protocol behaviors that may be interdependent. Moreover, in the embedded real-time systems where safety is considered, the composition can be even more complicated. This problem is addressed in [25], which provides an automated approach for synthesizing component behavior based on real-time coordination pattern that describe the behavior of connected component interaction. Each component participating in a coordination pattern has a role described by a protocol statechart. In order to synthesize the component external behavior the user needs to define a set of composition rules that explicitly describe the dependencies of components roles. This approach was implemented in the FUJABA Tool Suite [26]. The focus of this approach is on the external behavior of the components. However, in our case we will consider the conformance between the component internal behavior and all its ports, and will also check the compatibility between provided and required interfaces of the connected components.

WEAVER [27] is an Aspect-Oriented modeling tool developed by Motorola, which uses Specification and Description Language (SDL) to model the behavior due to its unambiguous semantics. WEAVER supports model execution, code generation and dependencies between aspects.

The CHARMY framework [28] uses model-checking techniques for validating software architecture conformance. An UML-like notation is used and the tool automates the validation. Static views show the component and connector relationships, while dynamic views describe the internal component behavior. The tool transforms these models into Promela code. To test a specific scenario described by a sequence diagram showing the exchanged messages between

components, the tool transforms the sequence diagram to Buchi Automata. Both the Promela code and Buchi Automata will be passed to SPIN to identify potential deadlocks.

6 Conclusions and Future Work

The paper introduces Component Erroneous Behavior Modeling (CeBAM), an aspect-oriented approach which provides a practical solution for modeling component erroneous behavior and error propagation. We illustrate, with the help of a case study, how to apply aspect-oriented techniques to model the erroneous behavior separately and then to compose automatically erroneous and normal behavior for each component. CeBAM is a part of a larger framework aiming to provide developers with automated tools for assessing the reliability (availability) of different fault tolerance mechanisms applied to a system under development. The composed state machines that are an outcome of CeBAM are used for conformance validation and for generating SRN models for dependability analysis.

We are working now on conformance validation of port behavior with internal behavior. Moreover, we are in the process of developing QVT transformations to generate automatically SRN analysis models from the software model. This transformation will be used for conformance validation and dependability attribute assessment. We are also investigating how to limit the state explosion in the analysis model without affecting too much the reliability and availability prediction results.

Acknowledgments. Naif A. Mokhayesh Alzahrani would like to acknowledge the support provided by Albaha University, KSA as well as the Ministry of Higher Education, KSA. This research was partially supported by the Natural Sciences and Engineering Research Council (NSERC).

References

1. Bernardi, S., Merseguer, J., Petriu, D.C.: Dependability modeling and analysis of software systems specified with UML. ACM Computing Surveys (CSUR) 45(1), Art. 2 (2012)
2. Immonen, A., Niemelä, E.: Survey of reliability and availability prediction methods from the viewpoint of software architecture. Software & System Modeling 7(1), 49–65 (2008)
3. Aysan, H., Punnekkat, S., Dobrin, R.: Error Modeling in Dependable Component-Based Systems. In: Proceedings of the 2008 32nd Annual IEEE International Computer Software and Applications Conference (COMSAC 2008), pp.1309–1314. IEEE Computer Society (2008)
4. Popic, P., Desovski, D., Abdelmoez, W., Cukic, B.: Error propagation in the reliability analysis of component based systems. In: Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering (ISSRE 2005), pp. 53–62. IEEE Computer Society (2005)

5. Cortellessa, V., Grassi, V.: A Modeling Approach to Analyze the Impact of Error Propagation on Reliability of Component-Based Systems. In: Schmidt, H.W., Crnković, I., Heineman, G.T., Stafford, J.A. (eds.) CBSE 2007. LNCS, vol. 4608, pp. 140–156. Springer, Heidelberg (2007)
6. Yedduladoddi, R.: Aspect oriented software development: an approach to composing UML design models. VDM Publishing (2009)
7. Bernardi, S., Merseguer, J., Petriu, D.C.: A dependability profile within MARTE. Software & Systems Modeling 10(3), 313–336 (2011)
8. Mencl, V.: Specifying component behavior with port state machines. Electronic Notes in Theoretical Computer Science 101, 129–153 (2004)
9. Moffett, Y., Beaulieu, A., Dingel, J.: Verifying UML-RT protocol conformance using model checking. In: Whittle, J., Clark, T., Kühne, T. (eds.) MODELS 2011. LNCS, vol. 6981, pp. 410–424. Springer, Heidelberg (2011)
10. Craig, D.C., et al.: Compatibility of Software Components - Modeling and Verification. In: Proceedings of the International Conference on Dependability of Computer Systems (DEPCOS-RELCOMEX 2006), pp. 11–18. IEEE Computer Society (2006)
11. Montecchi, L., Lollini, P., Bondavalli, A.: Dependability Concerns in Model-Driven Engineering. In: Proceedings of the 2011 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW 2011), pp. 254–263. IEEE Computer Society (2011)
12. Muppala, J., Ciardo, G., Trivedi, K.S.: Stochastic reward nets for reliability prediction. Communications in Reliability, Maintainability and Serviceability, 9–20 (1994)
13. Brosch, F., Buhnova, B., Koziolek, H., Reussner, R.: Reliability prediction for fault-tolerant software architectures. In: Proceedings of the Joint ACM SIGSOFT Conference – QoSA and ACM SIGSOFT Symposium – ISARCS on Quality of Software Architectures – QoSA and Architecting Critical Systems (QoSA–ISARCS 2011), pp. 75–84. ACM (2011)
14. Object Management Group: Query View Transformation (QVT) v1.1 formal/2011-01-01, `http://www.omg.org/spec/QVT/`
15. Gomaa, H.: Software Modeling and Design. Cambridge University Press (2011)
16. Avizienis, A., Laprie, J.C., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing 1(1), 11–33 (2004)
17. Object Management Group: Unified Modeling Language (UML) - Superstructure v.2.4.1 formal/2011-08-06, `http://www.omg.org/spec/UML/2.4.1/`
18. Selic, B.: A systematic approach to domain-specific language design using UML. In: Proceedings of the 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC 2007), pp. 2–9. IEEE Computer Society (2007)
19. Ali, S., Briand, L.C., Hemmati, H.: Modeling robustness behavior using aspect-oriented modeling to support robustness testing of industrial systems. Software & Systems Modeling 11(4), 633–670 (2012)
20. Goseva-Popstojanova, K., Trivedi, K.S.: Architecture-based approach to reliability assessment of software systems. Performance Evaluation 45(2-3), 179–204 (2001)
21. Abdelmoez, W., et al.: Error propagation in software architectures. In: Proceedings of the Software Metrics 10th International Symposium (METRICS 2004), pp. 384–393. IEEE Computer Society (2004)

22. Reussner, R.H., Schmidt, H.W., Poernomo, I.H.: Reliability prediction for component-based software architectures. Journal of Systems and Software 66(3), 241–252 (2003)
23. Cheung, L., Roshandel, R., Medvidovic, N., Golubchik, L.: Early prediction of software component reliability. In: Proceedings of the 30th International Conference on Software Engineering (ICSE 2008), pp. 111–120. ACM (2008)
24. Såtten, V., Kraemer, F.A., Herrmann, P.: Towards automatic generation of formal specifications to validate and verify reliable distributed systems – A method exemplified by an industrial case study. ACM SIGPLAN Notices - GCPE 2011 47(3), 147–156 (2012)
25. Eckardt, T., Henkler, S.: Synthesis of Component Behavior. In: Proceedings of the 7th International Fujaba Days, Eindhoven University of Technology, The Netherlands, pp. 40–44 (2009)
26. FUJABA Tool Suite, `http://www.fujaba.de`
27. Cottenier, T., Van Den Berg, A., Elrad, T.: Motorola WEAVR: Aspect orientation and model-driven engineering. Journal of Object Technology, 51–88 (2007)
28. Inverardi, P., et al.: CHARMY – An extensible tool for architectural analysis. In: Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE-13), pp. 111–114. ACM (2005)