Reliability Prediction in Model-Driven Development

Genaína N. Rodrigues[1], David S. Rosenblum[1], and Sebastian Uchitel[2]

[1] London Software Systems
Department of Computer Science
University College London
Gower Street
London WC1E 6BT
United Kingdom
{g.rodrigues,d.rosenblum}@cs.ucl.ac.uk
[2] Department of Computing
Imperial College London
180 Queen's Gate
London SW7 2RH
United Kingdom
su2@doc.ic.ac.uk

Abstract. Evaluating the implications of an architecture design early in the software development lifecycle is important in order to reduce costs of development. Reliability is an important concern with regard to the correct delivery of software system service. Recently, the UML Profile for Modeling Quality of Service has defined a set of UML extensions to represent dependability concerns (including reliability) and other non-functional requirements in early stages of the software development lifecycle. Our research has shown that these extensions are not comprehensive enough to support reliability analysis for model-driven software engineering, because the description of reliability characteristics in this profile lacks support for certain dynamic aspects that are essential in modeling reliability. In this work, we define a profile for reliability analysis by extending the UML 2.0 specification to support reliability prediction based on scenario specifications. A UML model specified using the profile is translated to a labelled transition system (LTS), which is used for automated reliability prediction and identification of implied scenarios; the results of this analysis are then fed back to the UML model. The result is a comprehensive framework for addressing software reliability modeling, including analysis and evolution of reliability predictions. We exemplify our approach using the Boiler System used in previous work and demonstrate how reliability analysis results can be integrated into UML models.

1 Introduction

The evaluation of system specifications early in the software development lifecycle has increasingly gained attention from the software engineering community. Early evaluation of software properties, including non-functional ones, is important in order to reduce costs in software development before resources have been allocated and decisions have been made. *Dependability* is one example of an important non-functional property and represents the ability to deliver service that justifiably can be trusted. One of the attributes encompassed by dependability is reliability, which is concerned with the correct delivery of software system service.

There has been growing interest in closing the gap between commercial design tools and quantitative evaluation of software systems. However, techniques available to validate a design against non-functional properties often require significant effort and expertise unrelated to the usual business of developing software. The UML 2.0 Specification itself has augmented the previous UML version so that software system characteristics, and in particular the dynamic aspects of software behaviour, can be represented more accurately [14]. As a result, mechanisms to represent various aspects of system design and analysis can be expressed within one consistent language for specifying, visualising, constructing and documenting the artifacts of software systems. As a result of the task force to make UML more comprehensive to cover all aspects and concerns of the software development lifecycle, UML extension mechanisms (particularly UML profiles) have been used to introduce capabilities for representing non-functional concerns in UML models [15, 16].

The UML Profile for Modeling Quality of Service and Fault-Tolerance (henceforth referred to as the QoS Profile) defines a set of UML extensions to represent dependability concerns (including reliability) and other non-functional requirements using the lightweight extension mechanisms of UML [15]. However, we believe the QoS Profile is not comprehensive enough to support reliability analysis, as it does not address the modeling of dynamic aspects (such as scenarios, component interactions, and operational profiles) often required in modeling reliability. On the other hand, dynamic aspects have been defined in the UML Profile for Schedulability, Performance and Time Specification (henceforth referred to as the SPT Profile), but they were not incorporated into the QoS Profile. Because a system consists of a set of interacting components such that the interactions can reveal faults [1], modeling and annotating these interactions appropriately can assist us in predicting software reliability.

In previous work, we defined a technique to predict software system reliability based on scenario specifications [19]. The technique relies on LTSA, the Labelled Transition Systems Analyser tool [21], which provides scenario-based model synthesis and model checking capabilities to support the analysis. In this work, we define a profile for reliability analysis by extending the UML 2.0 specification to support reliability prediction using our LTSA-based approach. Following this principle, our approach to metamodeling using the UML lightweight extension mechanisms (i.e., profiles) is consistent with the MDA white paper [11], which defines basic mechanisms to structure models consistently and to express formally the semantics of the model in a standardised way. The result is a framework for systematically and pragmatically addressing software reliability modeling, including reliability analysis and prediction, with analysis results integrated back with the UML modeling environment to support system reliability enhancement. We point out here that it is not our intent to propose a new, independent UML profile. To the contrary, our purpose is to contribute towards a more comprehensive profile for reliability modeling premised on existing directions sanctioned by the OMG [15]. We exemplify our approach using the Boiler System used in previous work and demonstrate how the analysis results can be applied back into the UML models.

This paper is structured as follows: In Section 2 we present the basic concepts related to our technique for software reliability prediction. In Section 3 we introduce our model-driven development framework for reliability prediction. In Section 4 we present the core steps of our MDA-compliant model-driven reliability prediction approach. We illustrate the application of our profile in Section 5. Related work is presented in Section 6, and we conclude in Section 7 with a discussion of future directions for our work.

2 Background

In this section we present a succinct description of our reliability prediction technique based on *scenario specification*, presented in detail in previous papers [19, 18].

Scenarios are partial descriptions of how components interact to provide system functionality. A scenario specification is formed by composing multiple scenarios possibly from different stakeholders. To support reliability prediction, we annotate a scenario specification with probability annotations and use LTSA to process the annotated scenarios. LTSA is a tool that allows using behaviour models of distributed systems as prototypes for exploring system behaviour, and for automated checking of model compliance to properties (i.e., model checking) [21].

2.1 Reliability Prediction

In Figure 1 we depict the major steps our reliability prediction approach comprises. The steps are applied to a scenario specification expressed as a collection of *Basic Mes-*

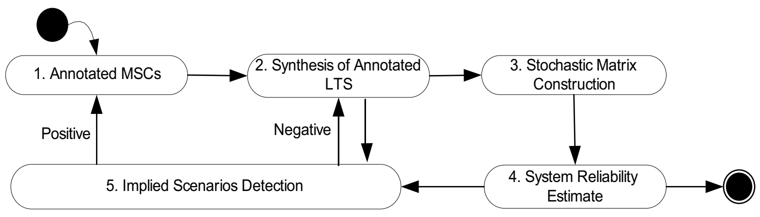

1. Annotated MSCs
2. Synthesis of Annotated LTS
3. Stochastic Matrix Construction
4. System Reliability Estimate
5. Implied Scenarios Detection
Positive
Negative

Fig. 1. The Steps of Our Reliability Prediction Approach.

sage Sequence Charts (henceforth BMSCs) and *High-Level Message Sequence Charts* (henceforth HMSCs). HMSCs provide sequential, conditional and iterative composition of BMSCs and other HSMCs, while BMSCs describe the message exchange between components on a time-line basis.

In the first step, we annotate the scenarios with two kinds of probabilities, *the probability of transitions between scenarios* PTS_{ij}, and *the reliability of the components* R_C. PTS_{ij} is the probability that the system will execute scenario S_j after executing scenario S_i. This information would be derived from an operational profile for the system [9] and is annotated on the HMSCs. The sum of the probabilities PTS_{ij} for all successor scenarios S_j must equal one. As for the component reliabilities R_C, they are annotated on the BMSCs. For the purposes of our approach, we interpret the reliability of a component C as being the probability of successful completion of an invocation of any service offered by C, irrespective of the execution time of the service.

The second step of our method is to synthesise a probabilistic Labelled Transition System (LTS) from the annotated scenario specification. This step is an extension of the synthesis approach of Uchitel et al. [22], in which a separate LTS is first synthesised for each component, and then the *system architecture* is taken as the parallel composition of the component LTSs. Our extension involves enhancements to this synthesis approach and exploits recent probabilistic extensions to the LTS formalism [2]. The enhancements have the effect of mapping the probability annotations of the scenario specification into probability weights for transitions in the synthesised architecture model. The probability weights of the composed LTS are computed according to the notion of *generative parallel composition* defined by D'Argenio et al. [5]. At the end of this step, it follows that for each state i of the synthesised architecture model and for all successor states j of i, $\sum_{j=1}^{n} PA_{ij} = 1$, where n is the number of states in the LTS architecture model and PA_{ij} is the probability of transition between state s_i and s_j of the composed LTS; $PA_{ij} = 0$ if the transition (s_i, s_j) does not exist.

In the third and fourth steps of our reliability prediction method, the architecture model synthesised in the second step is interpreted as a Markov model, and we apply the method of Cheung to compute the reliability prediction [3]. In particular, the probability weights of the architecture model are mapped into a square transition matrix whose row entries sum to one. To conform to Cheung's model, we extend the scenario specification to ensure that it contains exactly one initial and one final scenario. At this stage we can also perform sensitivity analysis of the prediction [18].

2.2 Implied Scenarios

Given a scenario specification, it may be impossible to build a set of components that communicate exclusively through the interfaces described and that exhibit only the specified traces when running in parallel [23]. The additional unspecified traces that are exhibited by the composed system are called *implied scenarios* and are the result of specifying the behaviour of the system from a global perspective while expecting the behaviour to be provided by components having only a local system view.

From the reliability prediction point of view, the existence of an implied scenario means that the system produces a trace that reveals a mismatch between behaviour and architecture. In that case, the model can exhibit behaviour (an implied scenario) that has not yet been validated and that, depending on whether it describes intended or unintended system behaviour, can impact system reliability. If we decide that the occurrence of the trace is desirable, we then need to appropriately place the *positive scenario* containing the trace into the scenario specification and annotate it with probabilities as described above. If we consider the occurrence of the trace as undesirable, then the scenario is a *negative scenario*, and the synthesised model must be constrained to prevent the occurrence of the negative scenario; this is accomplished by composing the synthesised model with an LTS that encodes the constraint [23]. We refer to the model where we apply such constraints as the *Constrained Model*, while the unconstrained model we refer to as the *Architecture Model*. In both cases, a new reliability prediction is computed from the revised model.

3 The Reliability Prediction Domain

Our framework for reliability prediction in model-driven development is based on the process depicted in Figure 2. The contribution presented in this paper is for steps 1, 2 and 3. The other steps constitute the work described in the previous section.

The framework consists of a UML profile for reliability prediction, plus a translation from the UML diagrams to LTSA. Reliability prediction is carried out as described before, as is the validation of the model by LTSA for implied scenarios. The result of this analysis provides a specification that has been elaborated through detection and validation of implied scenarios. Additionally, the results provide guidance to which software elements modeled in the UML profile the system is more sensitive. The rationale be-

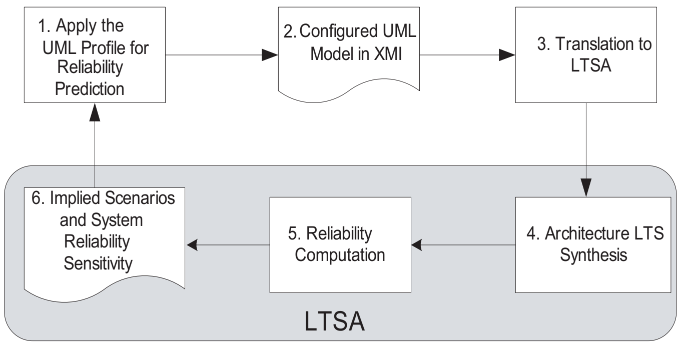

1. Apply the UML Profile for Reliability Prediction
2. Configured UML Model in XMI
3. Translation to LTSA
4. Architecture LTS Synthesis
5. Reliability Computation
6. Implied Scenarios and System Reliability Sensitivity
LTSA

Fig. 2. The Model Processing Framework for Reliability Prediction.

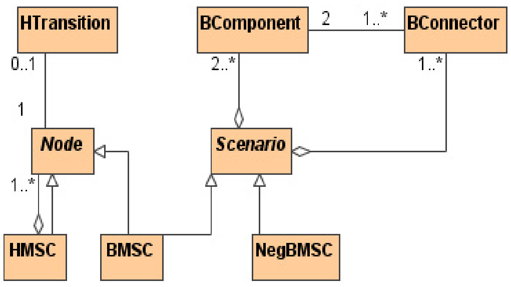


Fig. 3. The Domain Model of the Reliability Prediction Technique.

hind our approach is that the reliability of the system depends on two key pieces of information, as explained in Section 2: (1) scenario transition probabilities and (2) the reliability of the components. In order to support this approach within the MDA, we devise the conceptual model of reliability prediction depicted in Figure 3.

From Figure 3 it can be noticed that there are two main abstract constructs in our domain model: the *Node* and the *Scenario*. A *Node* represents the nodes of an *HMSC*. These nodes can be specialized as a *BMSC* or another *HMSC*, in case of a hierarchical HMSC. A *BMSC* corresponds to a Basic Message Sequence Chart describing the interactions between components participating in a scenario, and an *HMSC* corresponds to the high-level structure representing the composition of *BMSC*s. A *Node* element may be associated to an *HTransition* element, which represents the probability of transition to a node representing one of a set of alternative choices of behaviour. In that case, each node representing an alternatives is stereotyped as an *HTransition*.

A *Scenario*, the other main abstraction of our domain model, is an aggregation of at least two *BComponents* and at least one *BConnector*. A *BComponent* represents a software component, while a *BConnector* represents the logical or physical connection between two *BComponents*. The number of *BConnectors* associated to a *BComponent* is equal to the number of other components connected to that *BComponent*. A *Scenario* can be specialised as a *BMSC* or a *NegBMSC*, with the latter corresponding to a negative scenario as previously explained in Section 2.2.

In the next section, we delve more deeply into the processes depicted in Figure 2 and present the UML viewpoint of the structures in Figure 3.

4 Our Reliability Prediction Profile

Our profile for reliability prediction exploits the lightweight extension mechanisms of UML rather than the heavyweight mechanisms. Lightweight extensions of UML consist in defining a profile, i.e., a set of stereotypes, tagged values and OCL constraints. Heavyweight extensions of UML work in a higher level of abstraction by extending the Meta-Object Facility with new UML modelling constructs [13]. These extension features present in UML allow us to express the design and analysis domains seamlessly using the concepts inherent to these domains. Also, they permit us to map the behaviour of distributed component architectures into a domain representation preserving the semantics of UML in accordance with the MDA.

4.1 The UML Viewpoint

From the UML point of view, our profile depends on two major packages: (1) the SPT Profile, which defines the notion of time and resources modeling, and (2) the UML 2.0 Specification, where we realise the structures defined in the SPT domain and those required to model reliability.

In Figure 4 we show how the elements of our domain model relate to the elements that constitute the SPT Profile. The elements in Figure 4 in italicised font are part of the SPT Profile. A *Scenario*, in the SPT Profile, is an ordered series of steps called *action executions*, and a step, at one level of abstraction, can be decomposed further into a set of finer-grained steps. As can be noticed, all the elements in our domain model, except for *Node* and *Scenario*, extend elements of the SPT Profile. The abstract element *Node* of our domain, depicted in Figure 3, can be represented as a *Scenario* in the SPT profile or as coarse-grained *ActionExecutions*. A *Scenario* of our domain, depicted in Figure 3, can be represented as fine-grained *ActionExecutions* in the SPT profile. A *Scenario* in the SPT profile is specialized as an *HMSC* and an *ActionExecution* class is specialised as a *BMSC* or as an *HTransition*. The specialisation of the *ActionExecution* as an *HTransition* happens whenever the *ActionExecution* represents a choice of behavior. The *HTransition* also holds an association with *Resource Service Instance*, meaning that an *HTransition* keeps the reference of the resource service.

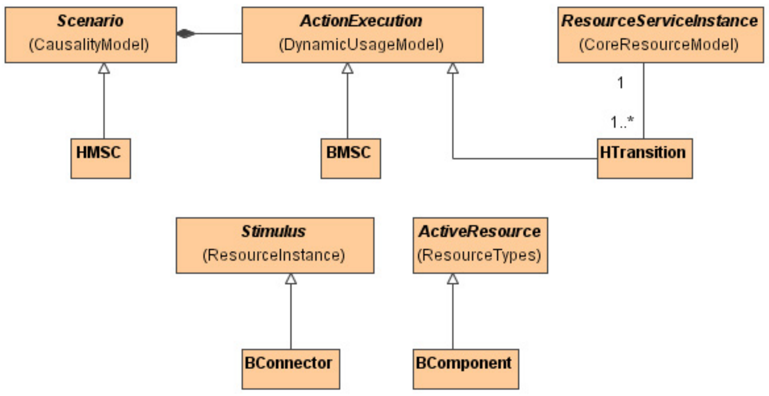


Fig. 4. Relationship between our Reliability Profile and the SPT Profile.

In SPT, resources are categorised being *passive* or *active*. Passive resources cannot generate their own behaviour, but only react to the occurrence of a stimulus, while active resources are those capable of spontaneous unprompted behaviour. The *BConnector* is a kind of passive resource, while the *BComponent* is an active resource.

Table 1 describes the elements constituting the UML profile for reliability modeling.

Table 1. Stereotypes and Tag Definitions for the Reliability Profile.

Stereotype	*Base Class*	*Tags*
≪ *HMSC* ≫	Interaction Activities	HName
≪ *BMSC* ≫	Interaction	BName
≪ *NegBMSC* ≫	Interaction	BName
≪ *HTransition* ≫	Interaction	PTS
≪ *BComponent* ≫	Classifier Component Instance	BCompRel
≪ *BConnector* ≫	Stimulus Message	BConnRel
≪ *Stop* ≫	Interaction	N/A

Tag	*Type*	*Multiplicity*
PTS	Real (0,1]	[0..1]
BCompRel	Real (0,1)	[0..1]
BConnRel	Real (0,1)	[0..*]
HName	String	[0..1]
BName	String	[0..1]

The stereotypes our profile comprises correspond to the concrete classes of our domain model depicted in Figure 3. Those stereotypes apply to UML 2.0 domain elements as follows:

– *BMSC* – Applies to *Interactions* of *Sequence Diagram* type.
– *NegBMSC* Applies to *Sequence Diagram*s with a *CombinedFragment* having *neg* as its *InteractionOperator*.
– *HMSC* – Applies to the *Interaction Overview Diagram*, which is the structure that best suits the modeling of an HMSC. *Interaction Overview Diagrams* focus on the overview of the flow of control where the nodes are *Interactions* or *InteractionOccurrences* [14]. Also, as a structure to represent the flow of control, the *Interaction Overview Diagram* enables the representation of the initial and the final states of the flow, which are also structures required in our reliability prediction technique [19]. Alternatively, we could use the *CombinedFragments* structure, but an *Interaction Overview Diagram* is semantically closer to HMSCs.
– *HTransition* – Applies to an *Interaction* representing an alternative choice of behaviour. It is tagged with the value *PTS*, the probability of transition to the *Interaction*.
– *BComponent* – Applies to components participating in *Sequence Diagrams* to be analysed by the model processor. The tag *BCompRel* associated to the *BComponent* stereotype represents the reliability of the component, as defined in Section 2.
– *BConnector* – Applies to messages exchanged between two *BComponents* in an *Interaction*. The tag *BConnRel* associated to the *BConnector* represents the reliability of the connector enabling the communication between the components. The reliability of the connector is regarded as the probability of success of a message transition, irrespective of the transition execution time.
– *Stop* – Due to the assumption in our prediction technique that there must be one final scenario in the scenario specification [19], it is required that no more than one *Interaction* connects to the *final node* of the HMSC *Interaction Overview Diagram*. The *Stop* stereotype applies to the *Interaction* with that feature.

The following are constraints defined in our Reliability Profile package:

1. Every *HMSC* and *BMSC* must be uniquely named.
2. Within an *Interaction Overview Diagram* stereotyped as an *HMSC*, every node must be either a *BMSC* or another *HMSC*.
3. Every *BMSC* is an *Interaction* of type *Sequence Diagram*.
4. Every *HMSC* must have one *Activity initial node* and one *Activity final node*.
5. *HMSC* nodes must have at least one incoming and one outgoing transition, except the *initial node* and *final node*.
6. The *PTS* values of *HTransition*-stereotyped nodes connected to the same *Decision* node within an *HMSC* must sum to one.
7. In an *HMSC*, there must be one (and only one) *Interaction* stereotyped as *Stop* and connecting to the *final node*.

Each of these constraints can be expressed easily in OCL, but for space reasons we do not present their OCL rendition.

4.2 Mapping from UML to LTSA

Once our profile is applied to a UML model, the translation from UML to LTSA is carried out. The transformation consists of (1) parsing the XML Metadata Interchange (XMI) form of the UML model, which is the standard representation of UML models in XML [12], and (2) generating the XML input format accepted by LTSA.

Current UML tools provide only partial conformance with the UML 2.0 specification, which has forced us to make some workarounds in our implementation. The major problem we encountered was to apply the stereotype *HTransition* and its *PTS* tagged value to the nodes (i.e., *Interaction Occurrences*) within *Combined Fragments* within *Interaction Overview Diagrams*. To get around this problem, we had to associate the *HTransition* stereotype with the transitions between nodes rather than to the nodes themselves. This solution is temporary, and we will evolve the implementation of our profile as tool support improves to properly accommodate the UML 2.0 specification.

We implemented the transformation of our UML profile to LTSA in XSLT [24]. XSLT describes rules for transforming a source document in a tree format (such as an XML file) into a result document described also by a tree. It therefore suits our need to transform the XMI representation of a UML model into the XML format accepted by LTSA. The transformation process is rather straightforward as long as the following conditions are satisfied:

1. An HMSC in LTSA cannot have multiple nodes that correspond to the same BMSC. In case there are multiple *Interaction Occurrences* of the same *Sequence Diagram* in a UML *Interaction Overview Diagram*, those multiple occurrences are reduced to just one node of the LTSA HMSC during the transformation process, keeping the same set of transitions contained in the *Interaction Overview Diagram*.
2. LTSA does not support hierarchically nested HMSCs at the moment. In case an *Interaction Overview Diagram* is specified in multiple hierarchical levels, it should be flattened before transformation is carried out.

4.3 Mapping Analysis Results Back to UML

After analysis has been carried out in LTSA following the approach presented in Section 2, we have the system reliability prediction and the detection of implied scenarios. In particular, we can use this analysis to provide answers to the following questions: Do we have any implied scenarios in our system architecture model? What is the impact of the implied scenarios on the system reliability? What is the sensitivity of the system reliability to changes in individual probability values?

If an implied scenario is a positive scenario, which means that the detected trace is to be included in the scenario specification, then a new Sequence Diagram is constructed for the trace and annotated with our profile for reliability prediction. This new *Interaction* is then incorporated appropriately as a node in the *Interaction Overview Diagram*. Incoming and outgoing transitions must be manually attached to the new positive scenario. If an implied scenario is a negative scenario, i.e., a trace to be avoided, it needs to be incorporated into a *NegBMSC*, with the undesirable message traces specified inside an *Interaction Fragment* having *InteractionOperator* type *neg*.

As for the sensitivity analysis, the purpose is to study the impact of components and usage profiles on system reliability [18]. For this purpose, the analysis reveals how the system reliability is sensitive to (1) the *component reliabilities*, and (2) the *scenario transition probabilities*. These two analyses can help in identifying components and scenario transitions that could threaten the reliability of the software system. The results produced by the sensitivity analysis can then be used by system designers to decide on mechanisms to use for enhancing the system reliability.

5 Example

We exemplify our approach using a variant of the Boiler Control system presented by Uchitel et al. [23]. As shown in the Interaction Overview Diagram of Figure 5, the Boiler Control system composes five Sequence Diagrams *Initialise*, *Register*, *Analyse*, *Terminate* and *Shutdown*, which are are depicted in Figure 6.

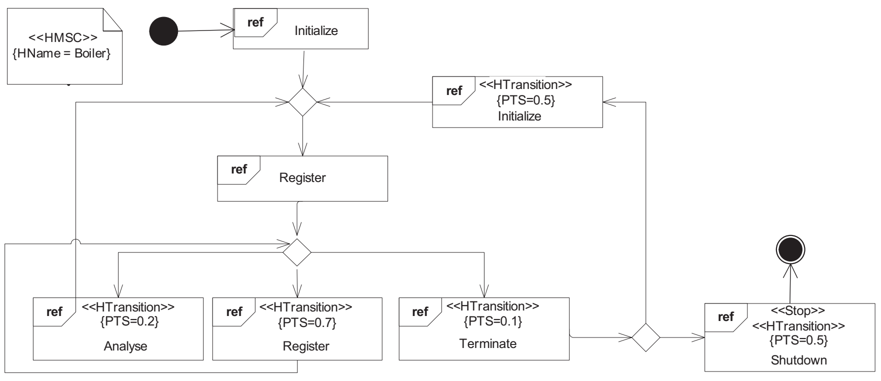


Fig. 5. The Interaction Overview Diagram of the Boiler System.

As presented in Section 4, the stereotype *HTransition* is tagged with the probability of transition between scenarios, *PTS*, as shown in Figure 5. The values for the *PTS* are based on the assumption that the system executes the scenario *Register* (which causes sensor readings to be entered into the database) far more frequently than the scenarios *Analyse* and *Terminate*, and that when it does execute *Terminate* there is an equal probability of either reinitialising or shutting down completely. As shown in the figure, it may be necessary to specify multiple references to the same Sequence Diagram if they are to be tagged with different scenario transition probabilities.

Inside the *BMSC*-stereotyped Sequence Diagrams, the components' reliabilities are annotated by applying the stereotype *BComponent* with its tagged value *BCompRel*, as depicted in Figure 6. Without loss of generality, we use coarse-grained, single values for the overall component reliabilities. In general, we can also associate finer-grained values for reliability through annotation of individual messages and segments of component timelines. The *BConnector* element of our profile suits the use of finer-grained

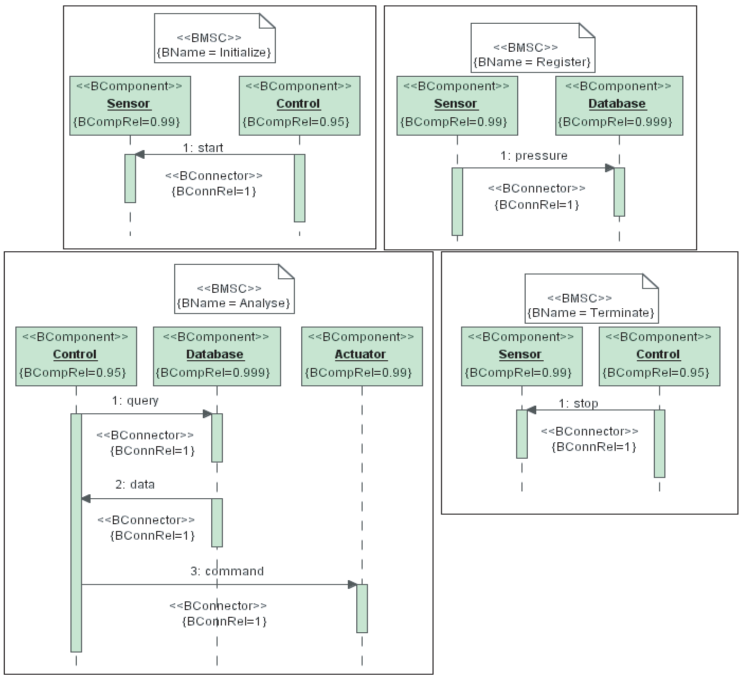


Fig. 6. The Annotated Sequence Diagrams of the Boiler System.

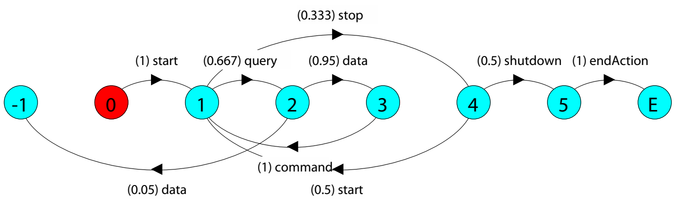


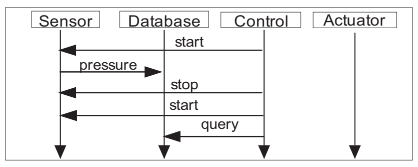
Fig. 7. The Synthesised Label Transition System for Component Control in LTSA.

values where individual messages can also be associated with a communication reliability value; in the example, these values are all set to 1.0. The values in Figure 6 for the reliability of the components reflect the assumption that the *Database* is a highly reliable, mature commercial software product, that the *Sensor* and *Actuator* are components whose hardware interface to the sensed/actuated phenomena will eventually wear out and fail, and that *Control* is a newer, complex software subsystem that still contains latent faults. Notice that the *Shutdown* Sequence Diagram is not present in Figure 6, as it has traces identical to those in the *Terminate* scenario.

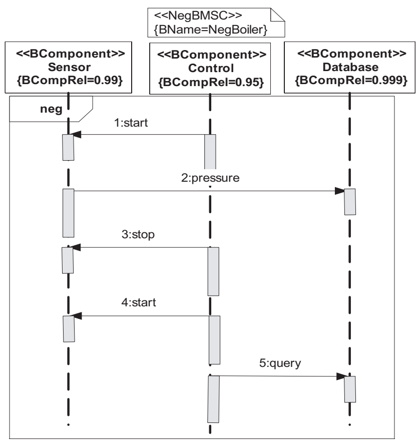
Following the steps of our reliability prediction technique [19], the LTS model for each component participating in the scenarios is generated; for instance, the LTS for component *Control* is depicted in Figure 7. Then, the *Architecture Model* of the system is synthesised as the parallel composition of the component LTSs. Using Cheung's approach to compute a prediction for the system reliability for the synthesised Architecture Model, we obtain a 64.9% probability of successful completion of the whole system execution, irrespective of time duration.

5.1 Validating for Implied Scenarios

The Boiler Control System specification of Figures 5 and 6, has implied scenarios, and Figure 8(a) depicts one of them. From the specification we see that the Boiler Control system architecture may exhibit the trace *start–pressure–query–data–command*, and that component *Control* interacts with *Database* only through messages *query* and *data*. However, in the implied scenario of Figure 8(a), a *query* is performed immediately after *start* but before a *pressure* reading is provided by the *Sensor* to the *Control*. In other words, the Architecture Model produces a trace that reveals a mismatch between behaviour and architecture, and we view this particular trace as being undesirable. This trace thus represents a negative scenario, and so a set of constraints preventing the occurrence of the negative scenario is expressed in FSP, the modeling notation of LTSA [7], and then composed with the Architecture Model. Following the steps of our



(a) Detected in LTSA



(b) Mapped to the UML Profile

Fig. 8. An Implied Scenario.

reliability prediction technique, a *Constrained Model* of the system is then synthesised as the parallel composition of the constraints with the Architecture model previously obtained. Calculating the reliability of the resulting Constrained Model, we obtain the value of 86.2% probability of successful completion of the whole system execution, irrespective of time duration. Figure 8(b) depicts the implied scenario detected in LTSA as it would be mapped back to UML as a *NegBMSC*.

5.2 Sensitivity Analysis

Sensitivity analysis consists of determining how the system reliability varies as a function of the components' reliabilities and scenario transition probabilities, with the purpose of identifying probabilities that have the greatest impact on the reliability of the software system [18]. For component reliabilities, the method consists of varying the reliability of one component at a time and fixing the others to 1. Then, computing the system reliability, we obtain the results presented in Figure 9.

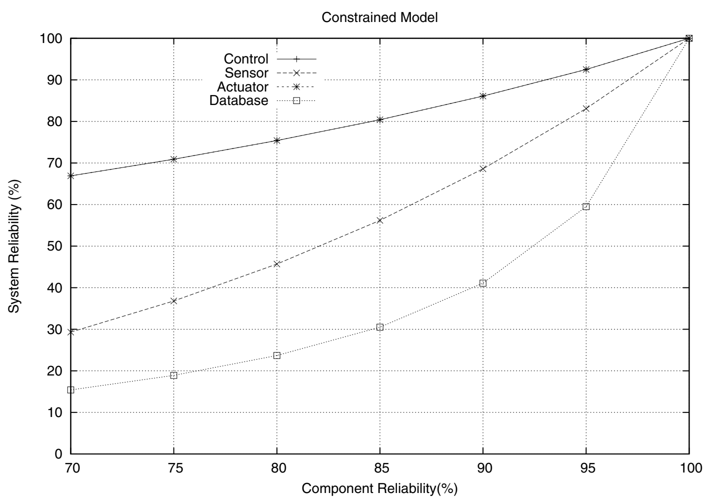


Fig. 9. The System Reliability as a Function of the Component Reliabilities.

The graph shows the system reliability of the Constrained Model as a function of the component reliabilities. The analysis shows that the reliability of the Boiler System is most sensitive to component *Database*, followed by components *Sensor*, *Control* and *Actuator*. Note that the *Control* and the *Actuator* curves coincide, meaning that they have an identical impact on system reliability.

6 Related Work

Using UML profiles to support modeling of non-functional aspects of software systems following a model-driven approach is not a new idea. The approaches for model-driven non-functional analysis are distinguished mostly by the way they support analysis of annotated UML models.

Majzik et.al. provide a profile for modeling fault-tolerant mechanisms, particularly redundancy, in UML diagrams [8]. Transformations are done in a sound manner through graph transformation, from UML to their analysis platform. Approaches in this category do not follow a standardised MDA approach. As a result, the key benefit of a standards-based approach is lost, i.e., interoperability of applications enabling a market in robust industrial tools that support the approach.

There has been work following the MDA approach for non-functional requirements modeling by extending the SPT Profile with regard to performance [6, 20]. Gu et.al. implement a transformation by parsing the XMI output of profile-mapped UML diagrams [6]. The approach of Skene et.al. resembles that of Gu et.al. but provides a more formal elaboration of the profile via OCL constraints [20]. Our approach follows in the same standards-based spirit, but with regard to reliability modeling. At the end of the day, any standards-compliant UML tool is capable of storing these models.

Recently, Cortellessa et.al. [4] proposed an amendment to the QoS Profile [15] with the purpose of addressing issues related to the reliability modeling of component-based systems. Our profile follows a similar structure as their extension for the QoS Profile, but we differ in the way we compose scenarios. In particular, we consider it important to provide more structure to a scenario specification and thus to model the interaction between scenarios through the *HMSC* structure of our profile. This feature allows us to model larger systems, as a greater number of scenarios can be analysed more easily through the *HMSC* structure. Therefore, we believe that our profile provides gains in modularity for modeling large systems and their reliability issues.

In our profile, we use of UML 2.0 constructs to support reliability analysis for component-based software systems. Constructs in UML 2.0 make easier the task of modeling non-functional requirements due to its richer expressiveness compared to previous UML versions. Reliability modeling using new concepts introduced in UML 2.0 are not commonly found in the literature. We believe that wider availability of modelling tools supporting UML 2.0 will stimulate new work in this area.

7 Conclusion

We present in this paper a UML profile to aid reliability prediction and analysis of software systems. We define a framework based on the UML 2.0 specification and the SPT Profile to support a reliability prediction technique that takes into account component structure exhibited in scenarios and the concurrent nature of component-based systems.

Following a compliant MDA process, a UML model specified using the profile is translated to a labelled transition system, which is model-checked by the LTSA tool to identify implied scenarios and is used to compute a reliability prediction according to the method of Cheung. Sensitivity analysis is also used to highlight components and scenario transitions that have a high impact on system reliability. The analysis results are integrated back with the UML modeling environment to support system reliability enhancement. Our purpose with the profile is to contribute towards a more comprehensive profile for reliability modeling consistent with the direction of the OMG [15].

We may acknowledge some shortcomings of our UML profile. We have implemented our profile using the MagicDraw UML tool version 9.0 Community Edition [10], and the XSLT stylesheet we implemented was based on the XMI version 1.2 the tool generates for the UML diagrams. The problem is that Magic Draw provides just partial support to UML 2.0, and the XMI output is out of date. Future versions of the tool are expected to be fully compliant with UML 2.0, as well as the XMI counterpart.

Future directions of our work include extending the profile to support modelling of fault-tolerance mechanisms. The first steps towards this goal were initiated in previous work [17] and by others in the literature, including the OMG itself [15]. By doing this, we intend to support code generation with assessed reliability, enhanced via fault-tolerance mechanisms present in current distributed component platforms. Additional work is also needed to explore methods and techniques that can fully reveal the impact of implied scenarios on system reliability. Finally, we plan to apply our approach on case studies of larger, more realistic systems to evaluate its scalability and the accuracy of the predictions it produces.

Acknowledgments

David Rosenblum holds a Wolfson Research Merit Award from the Royal Society. Sebastian Uchitel was partially funded by EPSRC grant READS GR/S03270/01. Genaína Rodrigues was funded by CAPES, grant 108201-9. Vittorio Cortellessa provided several valuable comments that improved our understanding of reliability modeling in UML and the state of current efforts with relevant UML profiles. We also thank the anonymous referees for their helpful suggestions on improving the manuscript.

References

[1] A. Avižienis, J. Laprie, and B. Randell. Fundamental Concepts of Dependability. In *Proc. IARP/IEEE-RAS Workshop on Robot Dependability*, May 2001.

[2] T. Ayles, A. Field, J. Magee, and A. Bennett. Adding Performance Evaluation to the LTSA Tool (Tool Demonstration). In *Proc. 13th Performance Tools*, September 2003.

[3] R. C. Cheung. A User-Oriented Software Reliability Model. In *IEEE Transactions on Software Engineering*, volume 6(2), pages 118–125. IEEE, Mar. 1980.

[4] V. Cortellessa and A. Pompei. Towards a UML profile for QoS: a contribution in the reliability domain. In *Proc. of the* 4^{th} *WOSP*, pages 197–206. ACM Press, 2004.

[5] P. R. D'Argenio, H. Hermanns, and J.-P. Katoen. On Generative Parallel Composition. In *Electronic Notes in Theoretical Computer Science*, volume 22. Elsevier, 2000.

[6] G. P. Gu and D. C. Petriu. Early Evaluation of Software Performance Based on the UML Performance Profile. In *Proc. of the 2003 CASCON*, pages 66–79. IBM Press, 2003.

[7] J. Magee and J. Kramer. *Concurrency: State Models and Java Programs*. Wiley, NY, 1999.

[8] I. Majzik, A. Pataricza, and A. Bondavalli. Stochastic Dependability Analysis of System Architecture Based on UML Models. In *Architecting Dependable Systems, LNCS–2667*, pages 219–244. Springer, 2003.

[9] J. D. Musa. Operational profiles in software-reliability engineering. *IEEE Software*, 10(2):14–32, 1993.

[10] NoMagic Inc. MagicDraw UML. http://www.magicdraw.com/.

[11] OMG. *Model Driven Architecture*. http://www.omg.org/mda/, July 2001.

[12] OMG. *XMI Specification*. http://www.omg.org/cgi-bin/doc?formal/2002-01-01, Jan 2002.

[13] OMG. *MOF 2.0 Specification*. http://www.omg.org/cgi-bin/doc?ptc/2003-10-04, Oct 2003.

[14] OMG. *UML 2.0 Superstructure*. http://www.omg.org/cgi-bin/doc?ptc/2004-10-02, 2003.

[15] OMG. *UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms*. http://www.omg.org/docs/ptc/04-09-01.pdf, Sep 2004.

[16] OMG. *UML Profile for Schedulability, Performance and Time Specification.* http://www.omg.org/technology/documents/formal/schedulability.htm, Jan 2005.

[17] G. Rodrigues, G. Roberts, and W. Emmerich. Reliability Support for the Model Driven Architecture. In *Architecting Dependable Systems II.* Springer, LNCS 3069, 2004.

[18] G. Rodrigues, D. Rosenblum, and S. Uchitel. Sensitivity Analysis for a Scenario-Based Reliability Prediction Model. In *Proc. ICSE 2005 Workshop on Architecting Dependable Systems*, pages 73–77, May 2005.

[19] G. Rodrigues, D. Rosenblum, and S. Uchitel. Using Scenarios to Predict the Reliability of Concurrent Component-Based Software Systems. In *Proc. ETAPS 2005 Conference on Formal Approaches to Software Engineering*, pages 111–126. Springer, LNCS 3442, 2005.

[20] J. Skene and W. Emmerich. A Model Driven Architecture Approach to Analysis of Non-Functional Properties of Software Architecture. In *Proc. of the* 18^{th} *ASE. Toronto, CA*. IEEE Computer Society, Oct. 2001.

[21] S. Uchitel, R. Chatley, J. Kramer, and J.Magee. LTSA-MSC: Tool Support for Behaviour Model Elaboration Using Implied Scenarios. In *Proc. of* 9^{th} *TACAS, Warsaw*, Apr. 2003.

[22] S. Uchitel, J. Kramer, and J.Magee. Synthesis of Behavioral Models from Scenarios. *IEEE Transactions on Software Engineering*, 29(2):99–115, Feb. 2003.

[23] S. Uchitel, J. Kramer, and J.Magee. Incremental Elaboration of Scenario-Based Specifications and Behavior Models Using Implied Scenarios. *ACM Transactions on Software Engineering and Methodologies*, 13(1):37–85, Jan. 2004.

[24] W3C. *XSL Transformations (XSLT).* http://www.w3.org/TR/xslt, November 1999.