1 A Model Driven Approach to Reliable Systems Provision

1.1 Short Introduction

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.

The main purpose of our work is to provide reliability for software systems following a model driven development approach. The model driven development has gained a significant attention from the software engineering community recently, but there are still many issues it has to tackle. Among those issues, reliability is still an open issue, and where I focus my current work.

In order to carry out reliability in the model driven development, I have devised some steps:

- Notation enhancements - using the lightweight mechanism of UML to create profiles with stereotypes, tagged values and constraints in order to express mechanisms that enhance the system reliability. [2]

- Model enhancements - the model representation.

- Analysis Technique (one family of analysis example) Throughout the process, we identified the need of an analysis technique: one that could be suitable for the early phase of software development. We realized that MDA needed to be extended to encompass software process. As a result, we propose a reliability analysis technique based on scenario specification, annotated with information derived from operational profiles: the components reliability and the transition probabilities between scenarios [5,4,3]. However there are some caveat to be considered before applying the model (see Section 1.2 based on the survey of [1]).

- Deployment configuration and code generation.

To understand how to accomplish the model driven reliability support, we depicted the overall process in Figure 1.
1.2 Model Assumptions, Limitations and Applicability

I found out this area I’m working is very interesting for the open questions I’ve found out. Many questions related to the architecture-based approach to quantitative assessment of software systems are still unanswered, and more research in this area is needed, in particular when it comes to the issues indicated below.

Level of Decomposition

Too many small components could lead to a large state space which may impose difficulties in measurements, parametrization, and of the model. However, too few components may cause the distinction of how different components contribute to the system failures to be lost. The level of decomposition clearly depends on the tradeoff between the number of components, their complexity and the available information about each component.

Estimation of Individual Components Reliabilities

Most of the work on architecture-based reliability estimation ignores the issue of how they can be determined. Assessing the reliability of individual components depends on the factors such as whether or not component code is available, how well the component has been tested, and whether it is a reused or a new component.

It is not always possible to use software reliability growth models for estimating the individual component’s reliability. A difficulty could arise due to the scarcity of failure data. Predictions based on failure data are of interest only if enough data are available in order to be statistically representative. Further diffi-
culties are due to the fact that the underlying assumptions of software reliability growth models such as random testing performed accordingly to the operational profile and the independence of successive testing runs can be seriously violated during the unit testing phase.

Another class of models estimate reliability from explicit consideration of non-failed executions. Such models can be used to make reliability estimations of the software based on the reliability of testing performed in its validation phase. In this context, testing is not a development activity for discovering faults, but an independent assessment of the software execution in representative operational environment. The problem here however is the number of executions necessary for estimating reliability levels which are commensurate with reasonable expectations.

Validity of Markov Assumption

All state-space models assume that the next component to be executed will depend probabilistically on the present component only and is independent of the past history. However, this needs to be justified wherever a Markov assumption is made.

If the probability to find the system in given state does not depend only on the last executed component but on a given number of components which were executed before the specified state was entered then the system can be described by a model of higher order. As a higher order Markov chain can be represented as a first order chain, consider a second order Markov chain. Let the pair of successive states \(i\) and \(j\) define a composite state \((i,j)\). Then the transition probability to composite state \((j,k)\) given the composite state \((i,j)\) is \(p_{ijk}\). This representation is useful because the results for the first order Markov chain can be carried over. Higher order Markov chain can be handled in a similar way. If the system can be described by a model of order \(r\) the state space is defined by a set of all possible \(r\)-tuples and the number of elements in the state space becomes \(n^r\).

The higher order Markov chain enables us to consider situations where execution time and failure behavior of components that are dependent on the number of components along the path from which the control was transferred. However, the size of the state space can grow so much that the model specification and analysis can become a difficult and error-prone process, as well as the measurements and parametrization.

Estimation of Transition Profiles

In architecture-based approach, there is a need to model the interaction of all components. In well designed system, interaction among components is limited.
In most cases, many interactions among components will be impossible, that is, transition probability matrix will be sparse. During the early phases of software life cycle, each component could be examined to find with which components it cannot interact. The remaining, non-zero transition probabilities may be available by analyzing program structure and using known operational profile. During the design phase, the estimation of transition probabilities may have to be done. During the integration phase, as new data become available, the estimates have to be updated to improve the predictions. The estimation of transition probabilities must be as accurate as possible since the error in this process will definitely affect system reliability and components sensitivities.

One feasible approach is to first obtain the estimates of the probabilities of execution of different scenarios are obtained based on the operational profile. Then, using analysis scenarios the transitions probabilities are calculated.

**Operational Profile**

Software reliability models assume that random test selection is guided accordingly to the user’s operational profile. Test selection aimed at finding faults, increasing various structure coverages or demonstrating different functional requirements are not representative of the user’s operational profile and might lead to reliability estimate different from one perceived by the user of the system. Also, upgrades to software might invalidate previous estimates of an operational profile. Therefore, it will be necessary to revise the architecture that describes component interaction.

Additionally, care must be taken to ensure that the change of operational profile is considered in assessing component’s reliability. When a reliability measure of a component developed for reuse is attached to the operational profile used during certification, one need to compare the certified operational profile with the environment where the component will be reused.

**Deriving the Model from Specification**

More recently there has been growing interest to close the gap between commercial design tools and quantitative evaluation of software systems. These efforts are mainly focused on the UML. UML provides system architects working on object oriented analysis and design with one consistent language for specifying, visualizing, constructing, and documenting the artifacts of software systems. Most of the work found in this regard consider the transformation of the UML specification into the stochastic performance models. The authors of the survey believe that the further advances of the methodological approaches for systematic and automatic generation of reliability and performance models from software
specifications will bring reliability and performance evaluation into the software design process.

1.3 Concerns About The Model

There are some concerns we are addressing and may result in further studies. This is the kind of excitement (and risk) research brings: it ends up revealing problems that leads the work to unpredicted directions. Some of the questions I have found include:

- Can mechanisms for Fault Tolerance assure reliability for software development? This question has been under quite hot discussion and John Knight, from University of Virginia is one of the main arguers.

- As far as the analysis part is concerned there are quite a lot of questions we are investigating, such as: how to make use of the information of the implied scenario into the design model to enhance reliability? I guess this question may be left for future work, which may lead to a post doc. What we can make use of the information from the analysis, so far, is: (1) which components have more impact on the overall system reliability, (2) what are the transitions that have more impact on the system reliability and (3) what is the overall behaviour of the system reliability. But how to know the probability that the implied scenarios will occur?

- What are the features to be used to enhance system reliability: replication, transaction management, message oriented communication, persistency,...? So far, replication seems the easier one to achieve. However, what is the compositional approach we are going to use: the formula of components in series(see our LNCS chapter in Architecting Dependable Systems) or the Fault-Tolerance architectural style of Wen-Li Wang et.al [6]? Using this approach, we will have to adjust the final architecture model generated from the Label Transition System Analyser Tool (LTSA). Designing the BMSCs (in UML terms is the Sequence Diagram) with replicas of the components, makes the analysis of the final model a bit hairy to look as the final model is parallel composition of the components behaviour and may increase the probability of having state explosion on the synthesized architecture model.

- Finally, the integration between the design and the analysis model. Use the XMI generated from the UML model and import it from LTSA, using the XML interface of the scenario specifications. The purpose is just the automation of the process. The first outcome of this work was recently published in our paper [3] at MoDELS (the former UML Series of Conferences)
References


