B. Regnell, P. Runeson and C. Wohlin, "Towards Integration of Use Case Modelling and Usage-Based Testing", Journal of Software and Systems, Vol. 50, No. 2, pp. 117-130, 2000.

Towards Integration of Use Case Modelling and Usage-Based Testing

Björn Regnell, Per Runeson and Claes Wohlin

Abstract

This paper focuses on usage modelling as a basis for both requirements engineering and testing, and investigates the possibility of integrating the two disciplines of use case modelling and statistical usage testing. The paper investigates the conceptual framework for each discipline, and discusses how they can be integrated to form a seamless transition from requirements models to test models for reliability certification. Two approaches for such an integration are identified: integration by model transformation and integration by model extension. The integration approaches are illustrated through an example, and advantages as well as disadvantages of each approach are discussed. Based on the fact that the two disciplines have models with common information and similar structure, it is argued that an integration may result in coordination benefits and reduced costs. Several areas of further research are identified.

1. Introduction

Over the last decades, much effort has been devoted to software implementation issues. However, as complexity grows, software development needs more than just programming. Design paradigms, such as object orientation, have entered the scene claiming to provide robust architectures and reusable components. Recently, the focus of the software research and practice has also approached issues related to requirements specification and reliability certification.

This paper presents recent research related to both these areas. The basic idea behind the presented work is to combine and integrate two different approaches that focus on the modelling of usage:

- **Use Case Modelling,** UCM, (Jacobson et al., 1992) and
- **Statistical Usage Testing**, SUT, (Mills et al., 1987).

Both UCM and SUT address phenomena related to the modelling of anticipated system usage, although with different background and terminology. UCM focuses on requirements analysis and usage modelling as a tool for describing and understanding requirements, while SUT focuses on usage modelling to enable test case generation for reliability estimation and certification.

In a survey of industrial software projects (Weidenhaupt et al., 1997), it is concluded that there is an industrial need to base system tests on use cases and scenarios. The studied projects, however, rarely satisfied this demand, as most projects lacked a systematic approach for defining test cases based on use cases.

UCM was introduced in the object-oriented paradigm (Jacobson et al., 1992) to complement traditional static object models with dynamic aspects. The work on use case modelling has in an object-oriented context primarily been focused on the transition from use case based requirements to high-level design, and how use cases can be used to find good object structures (Jacobson, 1995; Buhr and Casselman, 1996).

SUT, on the other hand, has focused on how to create a usage model that allows for generation of test suites which resemble operational conditions, by capturing the dynamic behaviour of the anticipated users. SUT research has concentrated on how such a model can be made scalable (Runeson and Wohlin, 1992; Wohlin and Runeson, 1994), but the main focus has been on the usage model itself from a testing perspective and not on the process of creating it from a requirements perspective. Hence, we see the need to study the common denominator of UCM and SUT in the perspective of requirements engineering, in search for an integrated framework for usage modelling.

There is an intimate relation between requirements specification and system validation; the major goal of validation is to show, for example through testing, that a system correctly fulfils its requirements. This fact is the main motivation behind combining and integrating use case modelling and statistical usage testing. Modelling effort related to the specification of system usage hopefully can be minimised if it can be used for both purposes (Wohlin et al., 1994). Software developers want to avoid modelling the same thing twice.

The main challenges in this work are:

- What concepts in use case modelling on the requirements level can be utilised in usage-based testing for reliability certification?
- How can we create a seamless transition between usage models for requirements specification and usage models for testing?

The presented work includes a conceptual study of each of the approaches, together with some preliminary results on how the approaches can be combined. Future work on an integrated usage modelling approach is also discussed. Chapter 2 presents the major motivations to integration and gives a general overview of integration approaches. Chapter 3 focuses on concepts in usage analysis for requirements specification with use cases, and Chapter 4 focuses on concepts in usage analysis for system validation with statistical usage testing. Chapter 5 presents two approaches to integrated usage modelling in more detail. Some conclusions are presented in Chapter 6.

2. An Integrated View on Usage Modelling

A schematic picture of the software development process is shown in Figure 1. The

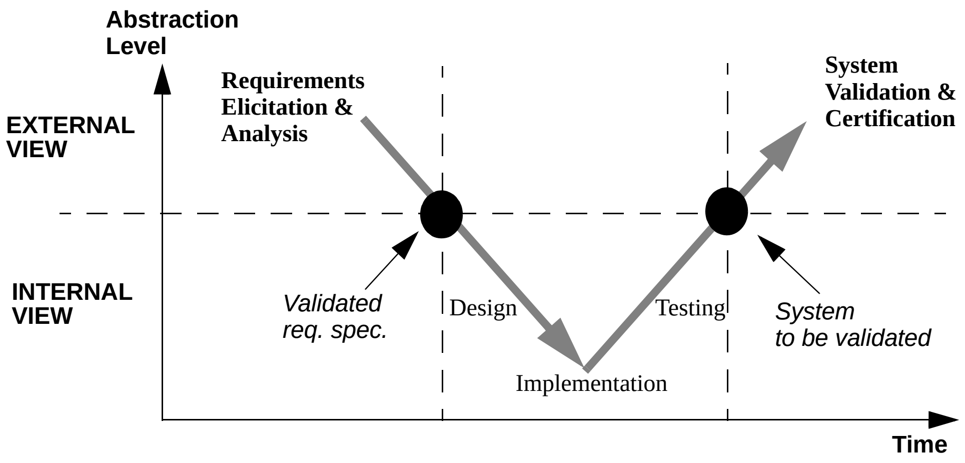


Figure 1. *Different abstraction levels over time.*

figure focuses on two dimensions: time and abstraction level. System development progresses from requirements elicitation and analysis, through design, implementation and testing, to validation and certification before delivery of a new system release. During this progress of activities, the focus changes with respect to abstraction. Development starts with an external view of the system, continues during design and implementation with a more detailed focus on system internal issues, before returning to an external “black-box” view. The external view focuses on externally observable functionality as seen by the users of the system, while the internal view focuses on system architecture and the structural and behavioural properties of objects within the system.

Usage modelling refers to the analysis and representation of system usage from the external viewpoint and thus fits well with both the early and late phases in Figure 1. This external view enables usage modelling on a high abstraction level, preventing models to be cluttered with internal details of the system.

Usage modelling deals with the dynamic relations between the events that take place when the system is used by its users. Thus, we are interested in stimuli to the system and its responses from the system actions issued by user stimuli. From a *requirements perspective* we want to capture what events should take place and in what order. Use case modelling, as presented in Chapter 3, is focused on the modelling of such dynamic aspects. A use case model describes a collection of use cases. Each use case covers a set of scenarios. The use cases determine the order of the events and define the possible alternatives in the flow of events.

When the system has been implemented, its function is determined by its program. System usage is, however, due to the free will of its human users, non-deterministic. The flow of events in system usage includes points where the next event is determined by a user action. A use case model represents the different usage possibilities and thus allows for non-deterministic choices of the users. From a *testing per-*

spective, we want to be able to capture this non-determinism in statistical terms by quantifying the probabilities (frequencies) of different alternative usages, in order to make the testing conditions resemble the foreseen operating conditions. Statistical usage testing, as presented in Chapter 4, is focused on both the usage dynamics and usage statistics, and quantifies the usage probabilities in, so called, usage profiles or operational profiles.

The statistical properties of system usage are also interesting from a requirements perspective. Information on, for example, how often a certain service is used is important input to the process of prioritizing requirements (Karlsson, et al., 1998). The probabilities of combining certain services are vital information when analysing how service combinations can interfere (Kimbler and Wohlin, 1995). Usage frequencies may also be used to optimize user interfaces.

Both Requirements Engineering (RE) and the Verification & Validation (V&V) have the same challenge of completeness and coverage. Have we covered all the essential requirements? Does the set of test cases cover adequately the requirements? Limited resources may require both requirements models and test models to be partial, giving the challenge of finding a level of coverage that is a good approximation of the complete system usage.

In summary, the *commonalities* between RE and V&V include:

1. Both areas desire models of system usage.
2. Both areas strive at an external view of the system.
3. Both areas benefit from quantification of usage frequencies.
4. Both areas have the challenge of adequate coverage.

Besides these commonalities, the motivation for integrating usage models in RE with usage models in V&V is based on the following *expected benefits*:

1. Modelling effort is reduced, as the same information is used for many purposes.
2. Traceability from requirements to test is promoted, which can be assumed to lead to less expensive maintenance.

Both RE and V&V have a variety of approaches proposed for usage modelling. In (Rolland et al., 1998) a survey of existing literature on scenarios and use cases in RE shows a great span of available methods. In (Jarke, et al., 1998) a survey of industrial practise revealed a great diversity in the ways that scenarios and use cases are applied. In (Graham, 1994) a number of different black-box testing techniques are outlined.

In this exploration of the possibilities of integration, we have chosen to focus only on two particular approaches: an extended version of Jacobson's use case modelling (Regnell, 1996), and an extended version of Whittaker's state-based markov model (Whittaker and Thomason, 1994; Runeson and Wohlin, 1995). These approaches are summarised in Chapter 3 and 4 respectively. Other specific approaches to use case modelling and usage-based testing can of course be combined in a number of ways. The specific integration approaches presented in Chapter 5 may be used as input to further research on the integration of other specific RE and V&V approaches to usage modelling.

In general, two different *integration strategies* can be identified, as illustrated in Figure 2.

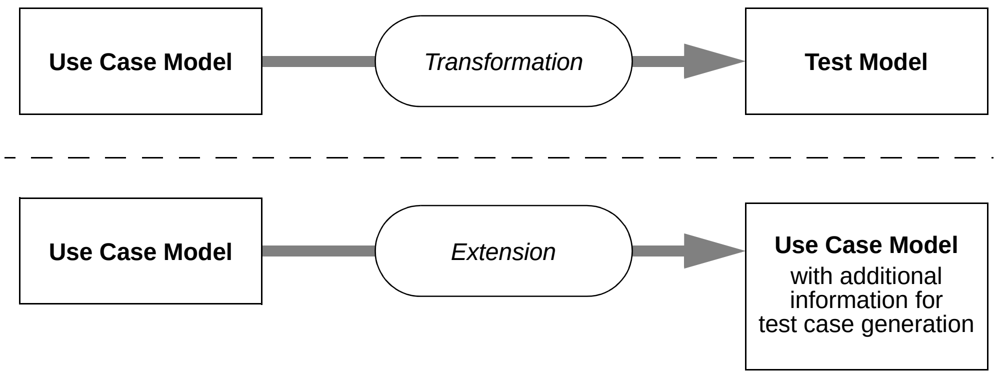

Use Case Model
Transformation
Test Model
Use Case Model
Extension
Use Case Model with additional information for test case generation

Figure 2. *Two ways of integrating usage modelling.*

The first integration strategy, **model transformation**, is based on the assumption that two different usage models are used: one for RE and one for V&V. The integration strategy requires guidelines for how to transform the information in the use case model in combination with additional test-specific information in the process of test model building.

The second integration strategy, **model extension**, is based on the assumption that a tailored use case model can be used directly for V&V, if it is extended with additional information necessary for testing.

The presented work represents an initial study on each of these strategies, but hard evidence on which strategy fits best with which context requires further research. There are, however, some basic reflections on the different strategies:

1. Model extension requires only one model, which can be assumed to imply less modelling effort and less expensive maintenance, compared to model transformation.

2. Model transformation may be more appropriate if the models differ greatly between requirements and test. By creating two models tailored for their special purposes, no compromise is needed. The common information may still be utilised through transformation rules.

Before we continue the investigation of the two proposed integration approaches, we present a conceptual study of each of the specific methods for use case modelling (Chapter 3) and statistical usage testing (Chapter 4) respectively.

3. Functional Requirements Specification with Use Cases

The elicitation, analysis and documentation of requirements on software systems is a crucial and non-trivial task (Loucopoulos and Karakostas, 1995; Bubenko, 1995). Well defined concepts and methods are needed when constructing specifications that represent requirements in an unambiguous, consistent, and complete manner. It is also important to have representations of requirements that are easily understood by the different stakeholders that take part in requirements analysis (Pohl, 1993). This Chapter concentrates on use case modelling for eliciting, analysing and documenting functional requirements. The use case concept has gained widespread acceptance within methods and notations such as OOSE (Jacobson, 1992), OMT (Rumbaugh et al., 1991; Rumbaugh 1994), the Booch method (Booch, 1994), ROOM (Selic et al., 1994), Fusion (Coleman, et al., 1994), and UML (Fowler and Scott, 1997).

There are many different possibilities of applying use cases and scenarios in requirements engineering. A survey of european software projects (Weidenhaupt, 1997) concluded that about two thirds of 15 visited projects used the OOSE (Jacobson, 1992) approach extended in various ways. Here we concentrate on one such extension (Regnell, 1996; Regnell et al., 1995; Regnell et al., 1996).

The main purpose of the presentation of the use case modelling concepts in this Chapter, and the statistical usage testing concepts in Chapter 4, is to provide a background to our objective of combining the two disciplines into an integrated framework, as discussed in Chapter 5.

In the subsequent sections, an example from the domain of telecommunication will be used as illustration. This example is a simplification of results from a case study conducted as a prestudy for the presented work, and involves a simplified Private Branch Exchange (PBX) with some common telephony services, such as unconditional call forwarding (CFU).

3.1 A Conceptual Framework for Use Case Modelling

The main idea behind use case modelling is to elicit and document requirements by discussing and defining specific contexts of system usage as they are anticipated by the different stakeholders in the requirements engineering process. The conceptual framework for the presented use case modelling approach (Regnell et al., 1996) and their relations are illustrated in Figure 3. These concepts are described in the subsequent sections.

Use cases can be viewed on different abstraction levels. At the **environment level**, the use case is related to the entities external to intended system. On this level, a use case is viewed as an entity representing a usage situation. At the **structure level**, the internal structure of a use case is revealed together with its different variants and parts. The **event level** represents a lower abstraction level where the individual events are characterized.

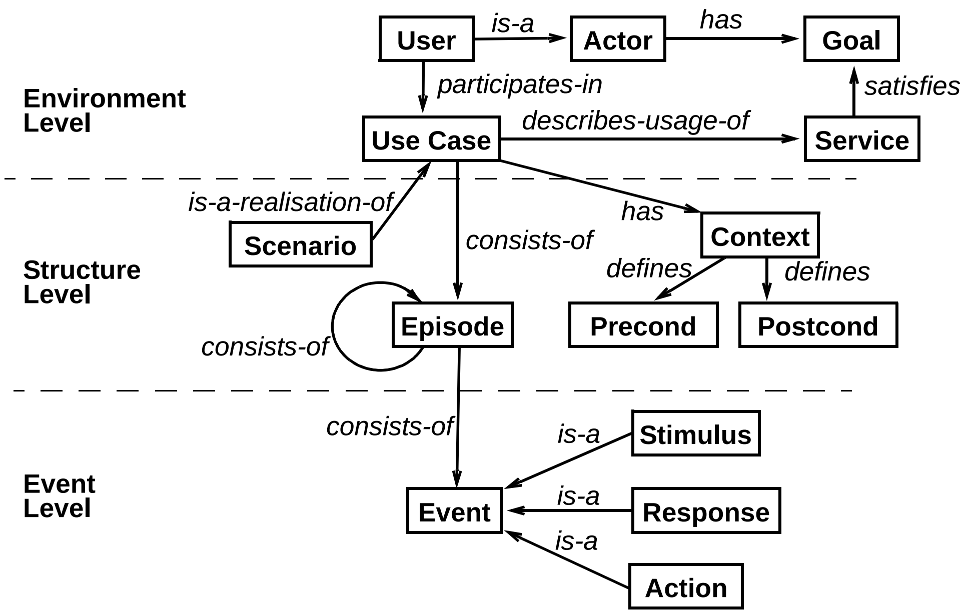


Figure 3. *Concept relations and levels of abstraction.*

3.2 Environment Level

The **users** belong to the intended target system's environment and can be either humans or other software/hardware based systems. Inside the target system we have a number of services. A **service** is a package of functional entities (features) offered to the users in order to satisfy one or more goals that the users have. Table 1, includes the services of our PBX example system.

Table 1. *Services in the PBX example.*

Service	Description
NCC	Normal Call with Charging
CFU	Call Forward Unconditional
RMR	Read Markings and Reset

Users can be of different types, called **actors**. A user is thus an instance of an actor. An actor (also called *user type* or *agent*) represents a set of users that have some common characteristics with respect to why and how they use the target system. Each actor has a set of **goals**, reflecting such common characteristics. Goals are objectives that users have when using the services of a target system. Thus, goals are used to categorize users into actors. Table 2 shows the goals of the two actors *subscriber* and *operator* in the PBX example.

Table 2. *Actors and their goals in the PBX example.*

Actors	**Goals**
Subscriber	GS1 To *achieve* communication with another subscriber
	GS2 To *cease* communication with another subscriber
	GS3 To *achieve* reachability at another destination
	GS4 To *cease* reachability at another destination
Operator	GO1 To *maintain* markings information representing call duration
	GO2 To *achieve* a printout of the number of markings for each subscriber
	GO3 To *achieve* a resetting of the number of markings for each subscriber

The goals are described as patterns using general temporal operators such as *achieve*, *cease*, and *maintain* (Dardenne and van Lamswerde, 1993).

A **use case** represents a usage situation where one or more services of the target system are used by one or more actors with the aim to accomplish one or more goals. Table 3 shows the use cases of the PBX example, and their relation to actors, goals, and services.

Table 3. *Uses cases in the PBX example.*

Use Cases	**Actors**	**Goals**	**Services**
Normal Call	Subscriber	GS1, GS2, GO1	NCC
Activate CFU	Subscriber	GS3	CFU
Deactivate CFU	Subscriber	GS4	CFU
CFU Call	Subscriber	GS1, GS2, GS3, GO1	CFU
Read Markings	Operator	GO2	RMR
Reset Markings	Operator	GO3	RMR

3.3 Structure Level

The structure level includes concepts that relates to the internal structure of use cases, such as different variants and parts of a use case.

A **scenario**[1] is a specific realisation of a use case described as a sequence of a finite number of events. A scenario may either model a successful or an unsuccessful accomplishment of one or more goals. A use case may cover an unlimited number of scenarios as it may include alternatives and repetitions. A scenario, however, is a spe-

1. Some authors use the terms scenarios and use cases as synonyms, but here we distinguish between them, to differentiate between type level and instance level.

cific and bound realisation of a use case, with all choices determined to one specific path. Table 4 shows a number of scenarios identified for the use case *normal call*.

Table 4. *Scenarios for the use case "Normal Call"*

Scenario	Description
Reply	Call to idle subscriber that replies
Busy Subscriber	Call to busy subscriber
No Reply	Call to idle subscriber that does not reply
Non-Existent	Call to non-existent subscriber
Timeout	Subscriber waits too long after offHook

The standardised language of Message Sequence Chart (MSC) (ITU-T, 1996; Regnell et al., 1996) may be used to express the structure level of a use case graphically. A High Level Message Sequence Chart (HMSC) is illustrated in Figure 4. Each scenario of the use case "normal call" is represented as an alternative.

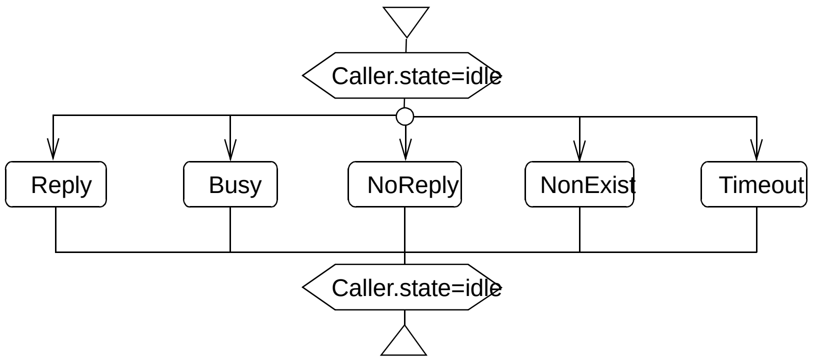


Figure 4. *High Level Message Sequence Chart.*

Each box with rounded corners refers to either another HMSC at a sub-structure level, or an MSC at the event level

Every use case (and scenario) has a **context** that demarcates the scope of the use case and defines its **pre-conditions** (properties of the environment and the target system that need to be fulfilled in order to invoke the use case) and **post-conditions** (properties of the environment and the target system at use case termination). An example of a pre-condition for the "CFU Call" use case is that "the CFU service has been activated". An example of a post-condition for the "normal call" use case is that "the caller is idle". Pre- and post-conditions are shown in Figure 4 as diamond symbols.

It is possible to have different degrees of scenario instantiation (Potts et al., 1994); a completely instantiated scenario corresponds to a system usage trace, where the sequence of events is totally ordered and every parameter has a specific value. A scenario may also be on a slightly higher level, having symbolic names instead of specific parameter values.

In use cases and scenarios it may be possible to identify coherent parts, called **episodes**. Similar event sequences may occur in several use cases, and episodes can be used as a modularisation mechanism to encapsulate use case parts and create a hierarchical use case model. We will not go into detail on episodes here, for more information see (Potts et al., 1994; Regnell et al., 1996).

3.4 Event Level

The lower abstraction level of uses cases, scenarios, and episodes includes **events** of three kinds: **stimuli** (messages from users to the target system), **responses** (messages from the target system to users), and **actions** (target system intrinsic events which are atomic in the sense that there is no communication between the target system and the users that participate in the use case).

Stimuli and responses can have **parameters** that carry data to and from the target system. In order to express parameters, and also conditions on data, a use case model may be complemented by a data model. A simple representation of a data model for the PBX example is given in Figure 5.

```
toneType =
 (dialTone, ringSignal, ringTone, busyTone, errorTone, infoTone);
maxNumberOfSubscribers: Natural;
SubscriberType: record (
    state: (idle, busy, off);
    telNumber: TelNbrType;
    CFU_active: Boolean;
    CFU_number: TelNbrType;
    markings: Natural;
    talkingTo: SubscriberType);
SubscriberSet:
    Set (1..maxNumberOfSubscribers) Of SubscriberType;
```

Figure 5. *A data model for the PBX example.*

Given a data model, we may express conditions on the data model that always should be true. Figure 6 shows such invariants for our PBX example.

When describing the information exchange between actors and the target system, it may be useful to define unique names for messages (stimuli and responses) together with information on the data types of their parameters. Table 5 presents the identified messages for our PBX example.

Based on the data model, the message definitions and natural language descriptions of scenarios, MSC can be used to describe graphically the event level as shown in Figure 7.

Each actor instance and the system are represented by a vertical time-axis, with time progressing downwards. Stimuli and responses are represented by arrows between actors and the system. Conditions are expressed as assertions on the data model in Figure 5.

It is also possible to describe alternatives and repetitions on the event level. Figure 8 shows a simple example of an alternative operator on the event level, expressing a choice between either stimuli A or response B. For more details on operators for ordering events, see (Regnell et al., 1996).

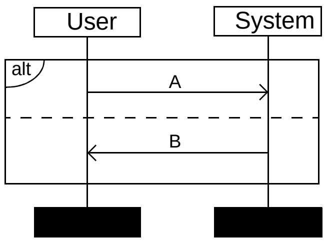


Figure 8. *Example of operator notation on the event level.*

4. Reliability Requirements Specification and Certification

Non-functional requirements are an essential part of requirements specifications. In particular, the reliability requirements are often regarded as one of the most important non-functional requirements. The reliability requirements cannot be formulated as a single figure (e.g. probability for failure-free execution or mean time between failures), since more information is needed. The reliability depends not only on system properties, for example correctness, but also on the system environment, i.e. how the

The telephone number is unique:

```
For-all X in SubscriberSet:
    For-all Y in SubscriberSet:
      if X<>Y then X.telNumber<>Y.telNumber;
```

If state is idle, the subscriber is not connected to another subscriber.

```
For-all X in SubscriberSet:
   (if X.state=idle then X.talkingTo=nil)
```

If subscriber X is talking to Y then subscriber Y is talking to X:

```
For-all X in SubscriberSet:
   if X.talkingTo <> nil then
      X. state = busy and
      Exist Y in Subscriberset:
         Y.state=busy and Y.talkingTo=X and
         X.talkingTo=Y;
```

Figure 6. *Some invariants in the PBX example.*

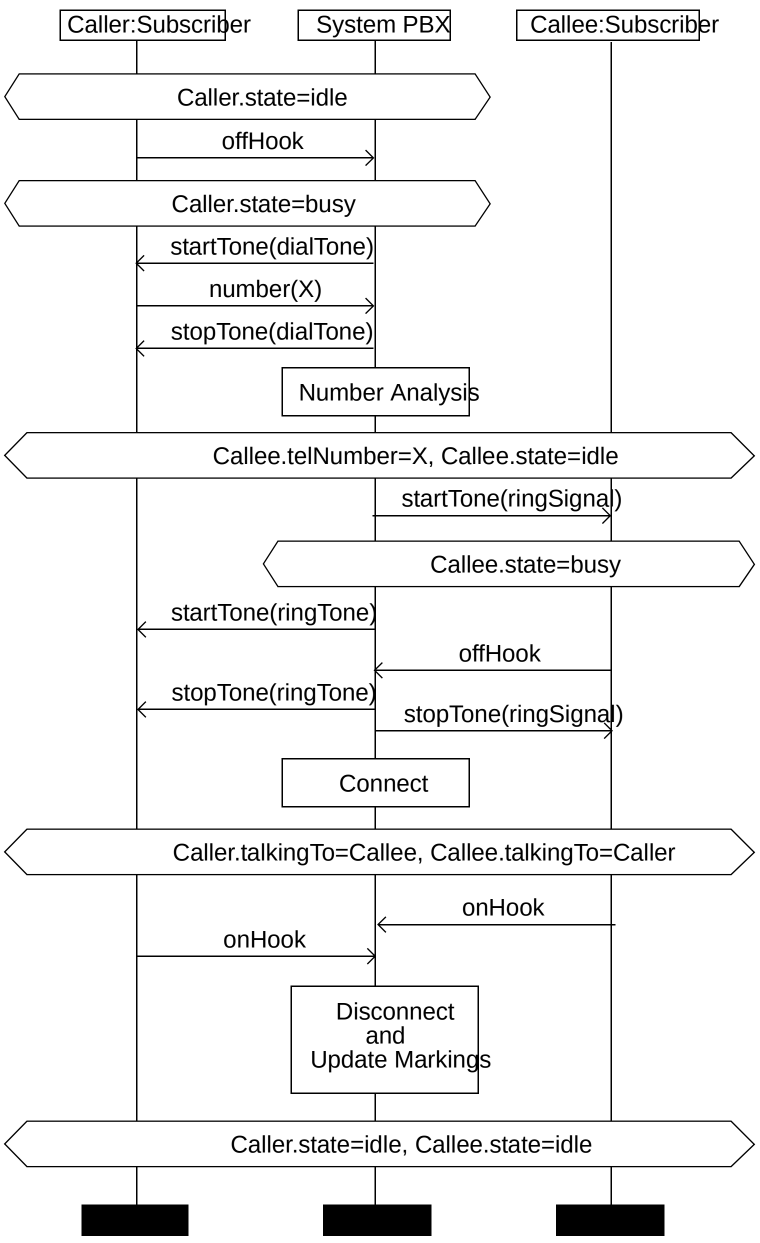


Figure 7. *MSC for the scenario "Reply" of use case "Normal Call".*

system is used. It is necessary to take the anticipated usage into account as the reliability of the system is dependent on the usage; the usage for which the requirement is valid must be stated together with the requirements of the system.

Usage-based testing with reliability certification is a means for validating reliability requirements. Functional requirements are validated at the same time. Thus, usage-based testing allows for both functional requirements validation and reliability certification.

This Chapter describes the concepts and representations of a particular usage-based testing approach, and Chapter 5 provides examples of usage-based testing

Table 5. *Messages in the PBX example.*

Message	**Description**
offHook	From Subscriber when lifting receiver
onHook	From Subscriber when hanging up the receiver
number(telNbrType)	From Subscriber when dialling a Number
activateCFU(telNbrType)	From Subscriber when activating CFU
deactivateCFU	From Subscriber when deactivating CFU
startTone(toneType)	To Subscriber when a tone is given
stopTone(toneType)	To Subscriber when a tone is stopped
readMarkings	From Operator when issuing a reading of markings
resetMarkings	From Operator when issuing a reset of markings
markings(markingListType)	To Operator when reporting the markings

models (using the PBX system of Chapter 3), and discusses how to integrate usage-based testing with use case modelling.

4.1 Usage-based Testing

There are two major approaches to testing: *black-box testing* and *white-box testing*. Black-box testing techniques take an external view of the system and test cases are generated without knowledge of the interior of the system. White-box testing techniques take an internal view and aim at covering all paths in the code or all lines in the code or maximising some other coverage measure. The main objective of all testing techniques is to validate that the system fulfils the requirements; mostly the focus is on functional requirements, but test cases can also address quality issues. For example, they can either be derived with the objective to locate as many faults as possible or to certify the reliability level of the software.

Usage-based testing implies a focus on detecting the faults that cause the most frequent failures, hence maximising the growth in reliability. This paper focuses on black-box testing and in particular on usage-based testing, which can be used to certify a particular reliability level and, of course, to validate the functional requirements.

The ability to certify software during testing is based on a user-oriented approach. This requires a model of the anticipated usage of the software and quantification of the expected usage as the software is released. Several approaches have been investigated and used in this area. Musa (1993), for example, advocates operational profile testing, Mills et al. (1987) discuss random testing based on the operational profile and Runeson and Wohlin (1992; 1995) present an approach with user-state dependent random testing based on the operational profile. The focus in this paper is on the latter approach, i.e. statistical usage testing based on a *state hierarchy model* (Wohlin and Runeson, 1994; Runeson and Wohlin, 1998). The subsequent sections present the conceptual framework for this approach and discuss how the concepts are represented in the state hierarchy model.

4.2 A Conceptual Framework for Usage-based Testing

A system consists of a number of services provided to the system users. These services are implemented by objects. The objective here is to provide a framework for modelling usage and to enable reliability certification of objects that are parts of a system, as well as certification of an entire system. In Figure 9, the relations between target system and environment concepts are illustrated. These concepts form the basis for creating a model of the usage of the certification object and also for quantifying the anticipated usage. The concepts of the usage specification test model are further discussed below and shown in Figure 10.

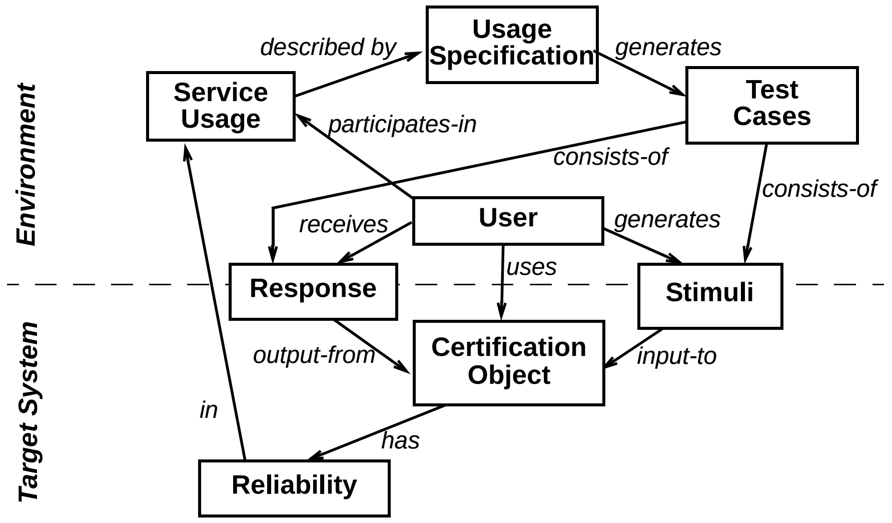


Figure 9. *Certification concepts and their relationships.*

The concepts in Figure 9 can be defined as follows. The software to be certified is referred to as a the **certification object**. A certification object has a certain **reliability**, which is the probability that the object works as intended for a specified time and in a specified **service usage** environment. The certification object is used by one or many users, which can be either human users or other systems. The communication between the user in the **environment** and the **certification object** is made through **stimuli** generated by the user and **responses** sent by the object. The user of the certification object participates in service usage, which is described by the **usage specification**. From the usage specification, **test cases** are generated including stimuli and responses to/from the certification object.

To enable certification, the environment must be modelled to allow for generation of test cases which resemble the anticipated behaviour in the operational phase. Thus, modelling concepts capturing the environment are needed. Depending on the type of testing being applied, different test models have to be derived. The focus here is on usage-based testing, which means that the test model is a **usage specification**, see Figure 10. The usage specification consists of a **usage model**, which describes the possible behaviour of the users, and the **usage profile**, which quantifies the actual usage in terms of probabilities for different user behaviour.

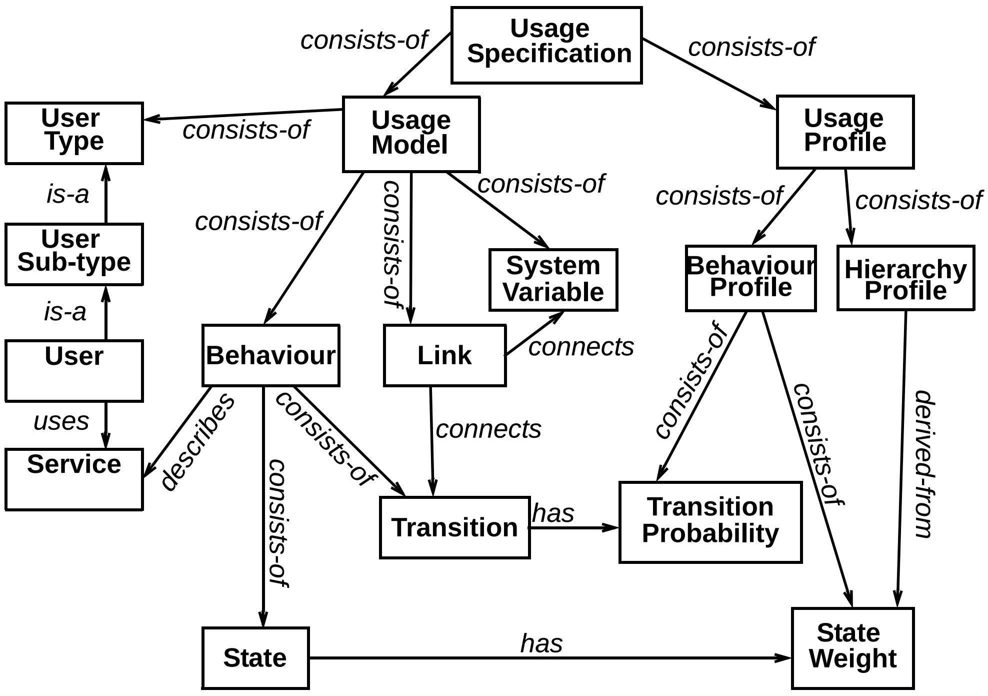


Figure 10. *The usage test model and its usage-oriented modelling concepts.*

The usage model is described through a hierarchy which defines the **users** and their relations (**user types** and **user sub-types**), the **behaviour** models which define the user **states** and **transitions** between user states, the **system variables** which capture important assets of the system state and the **links** which define connections between different behaviour models and system variables.

The usage profile is divided into a **hierarchy profile**, which describes the probabilities for choosing one specific user in the environment, and the **behaviour profile**, which models the behaviour of a single user, while using the available services.

The hierarchy is a tree structure where the nodes in the tree represent groups of users, based on their usage models and usage profiles. A **user type** is defined as the collection of users having the same possible behaviour (normally equivalent to that the users have the same goal, cf. actor), i.e. they have exactly the same behaviour models. A **user sub-type** is a further division of the users into a group where all users also have the same behaviour profile, hence having a similar statistical behaviour. The **users** are instances of a user sub-type and each user has access to a set of **services**, which usage is described by behaviour models.

4.3 Hierarchical Representation of User Behaviour and

Usage Profiles

The usage model has to two main parts, the hierarchy and the behaviour parts. The two model parts are illustrated through a small example in Figure 11 and Figure 12, which show a part of the PBX example.

For each instance of a service, there is a behaviour model, which consists of states and transitions. The services in Figure 11, has been divided into states, and the possible transitions among the states are also shown.

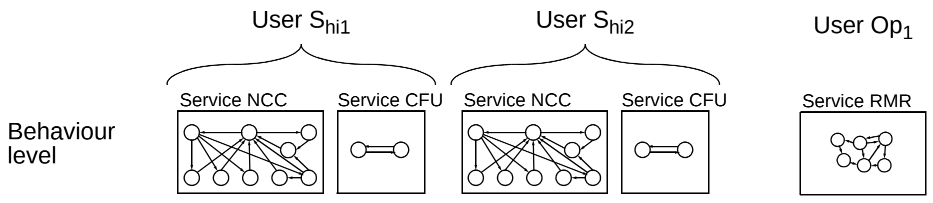


Figure 11. *Behaviour model for a part of the PBX example.*

It should be noted that the states are external states, i.e. **user states**, which are only a subset of the possible system states. User states describe the externally visible states of the system.

The total state of the usage model is a vector *T* of states for all behaviour models:

$$T_{\text{Behaviour model}} = \begin{bmatrix} \dots & t_{S_{hi1}\,NCC} & t_{S_{hi1}\,CFU} & t_{S_{hi2}\,NCC} & t_{S_{hi2}\,CFU} & t_{Op_1\,RMR} \end{bmatrix}$$

where t_{ij} means the state of service *j* for user *i*.

Probabilities are assigned to each arc in the instances of the behaviour models, hence taking different profiles into account. Furthermore, state weights are assigned to each state (denoted W_k for service *k*), which reflect the overall stimulus frequency of the user being in that state.

In Figure 12, the hierarchy is shown, which breaks down the usage of the certification object into individual users and their services.

Two user types have been identified, which implies, for example, that all users of user type *Subscriber* must have the same behaviour model. *Subscriber* is divided into two user sub-types, hence modelling differences in the behaviour profiles, i.e. users S_{lo1} through S_{lo5} do not have the same behaviour profiles as users S_{hi1} and S_{hi2}. These users use the services NCC and CFU.

The *Operator* user type only consists of one user sub-type Op, and only one user exists of this sub-type, i.e. user Op_1. This user uses a single service, i.e. service RMR. (This example is further elaborated in Section 5.1.)

The hierarchy profile is a little bit more complicated. The complication arises as it is reasonable to change the probabilities in the hierarchical profile based on the state of the users. The probability for the selection of a service (denoted p_k for service *k*) equals the current state weight of the service, divided by the sum of the current state weights for all services.

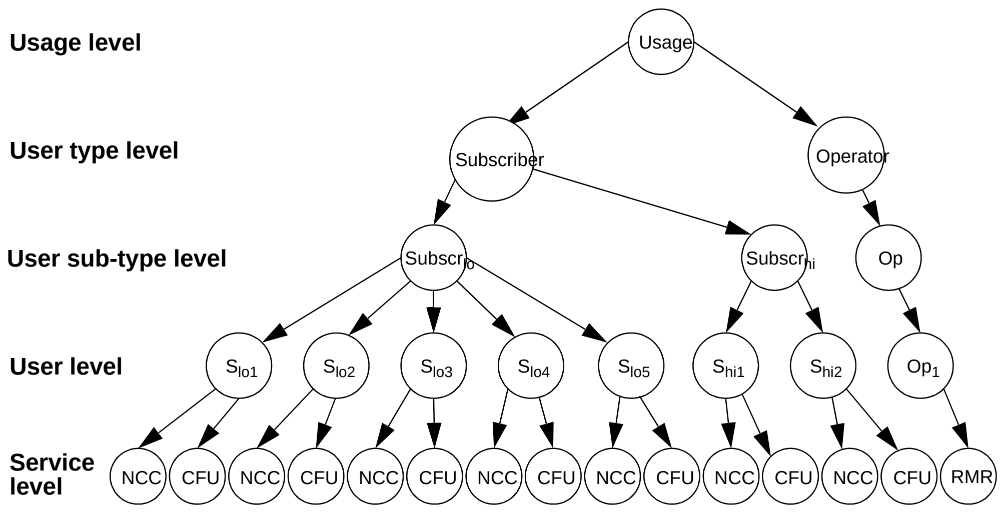


Figure 12. *The hierarchy part of the usage model, using the PBX example*

$$p_k = \frac{W_k}{\sum_j W_j} \text{ if at least one } W_j \neq 0$$

For each transition in a behaviour model, the hierarchy profile is updated. The update algorithm is described in more detail in (Runeson and Wohlin, 1998).

The usage specification is run through, using random numbers. To generate test cases, the tester is supposed to act as the system and provide the expected responses of the certification object using the requirements specification as a basis. The stimuli generated from the usage specification and the responses from the tester are stored on a test file. The test generation procedure is further described in (Wesslén and Wohlin, 1995).

5. Two Approaches to Integrated Usage Modelling

The combination of use cases and usage-based testing provides a comprehensive view of the software development process from a user perspective (Wohlin et al., 1994). The user does not have to bother about internal technical solutions. Instead the user can focus on the external view and the actual use of the system. However, to make the combination seamless, there is a need for bridging the conceptual gap between the two approaches. This Chapter presents two approaches to integration.

As stated in Chapter 1, use case modelling and usage based testing are sprung out of different traditions and have different objectives. It should be noted, however, that both usage-based testing and use case modelling need similar information, which means that the information is not collected solely for either requirements specification or testing purposes; use case models contain much information that can be used for system validation. Although there are many conceptual similarities between use case modelling, as presented in Chapter 3, and usage-based testing, as presented in Chapter 4, there does not exist a simple one-to-one mapping between the concepts in the two disciplines. The use case modelling concepts do, for example, not cover the stochastic semantics of usage profiles, and the state hierarchy model does, for example, not cover the concepts of actor goals or pre- and post-conditions.

When trying to combine the concepts of Section 3 and Section 4 to form an integrated approach to usage modelling, two integration approaches can be identified: (1) we could try to *transform* a use case model into a state hierarchy model by translating the concepts in the former to the concepts in the latter according to some concept mapping rules, or (2) we could try to *extend* the use case model to incorporate stochastic semantics and the necessary information for test case generation.

Section 5.1 and Section 5.2 proposes a working procedure for each of these integration approaches, and the PBX example presented in Section 3 is followed in both approaches. Section 5.3 discusses their advantages and disadvantages.

5.1 Approach 1: Integration by Transformation

The first approach to integration is based on the observation that many of the concepts in use case modelling and statistical usage testing have similar semantics. For such similar concepts it may be possible to use simple translation guidelines, and together with the necessary additional information on the stochastic properties of users we can create a state hierarchy model by transforming the use case model. This transformational approach is sketched in Figure 13.

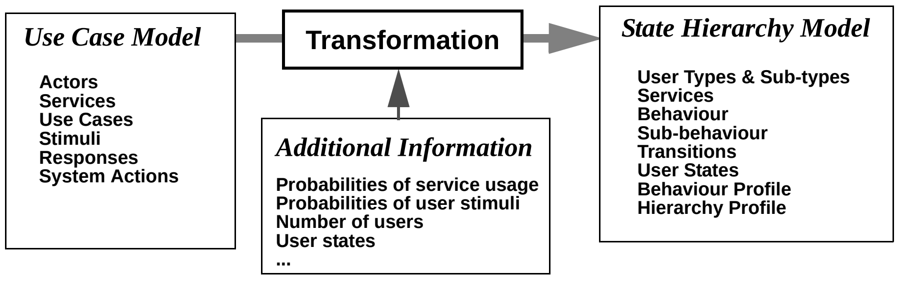


Figure 13. *Transforming a use case model into a state hierarchy model.*

In general, the transformation includes moving from an event based representation to a state-based representation and adding other necessary information.

When creating the state hierarchy model, it may be suitable to follow the method outlined below (Runeson and Wohlin, 1998). The transformation activity uses the concepts captured by the use case model and additional information of usage probabilities and quantities.

1. Identify services
2. Define user types
3. Define user sub-types and instantiate users
4. Create behaviour models
5. Define the behaviour usage profile
6. Define the hierarchy usage profile

This method shall not be seen as completely defined steps; iterations are performed when needed. Below the steps are presented.

Identify services. The services can be used directly as defined in the use case model, see Table 1. In our example the resulting service list is: NCC (Normal Call with Charging), CFU (Call Forward Unconditional) and RMR (Read Markings and Reset).

Define user types and sub-types. The actors in the use case model are the basis for the upper levels of the hierarchy model. Each actor becomes a user type and user sub-types are added if there are different probability profiles for a user type. Additional information on the estimated number of instances of each user sub-type determines the user level in the hierarchy model.

The example has two actors in the use case model, *Subscriber* and *Operator* (see Table 2), which constitute user types in the usage model. The *Subscriber* user type has access to two of the services, NCC and CFU, and the *Operator* user type has the RMR service.

In addition to the use case information, quantitative information is gathered from other sources. There are two variants of the Subscriber type, one with high frequency usage and one with low frequency; each constituting a user sub-type. For each of the user sub-types, the number of instances are defined as well. There are 5 low-frequency subscribers, 2 high-frequency subscribers and 1 operator. These steps result in the hierarchy model as presented in Figure 12.

Create behaviour and sub-behaviour models. The information for the behaviour models are not directly available in the use case model as the information for the hierarchy model are. However, there are parts of the information in the use cases and the scenarios (see Table 3 and Table 4 respectively) which can be integrated to a behaviour model for the service in question. Furthermore, the messages (see Table 5) constitute the interface between the system and its users and will hence appear in the usage model as well.

The behaviour model is a state-transition diagram in which use cases and scenarios constitute parts. The state information can in parts be collected from the pre- and postconditions for the use cases. The messages are attached to the transitions in the behaviour model, as stimuli to the system.

In our example, the NCC service behaviour model is further elaborated. The Normal Call use case and its five scenarios (Reply, Busy Subscriber, No Reply, Non-Existent and Timeout) is the starting point for the behaviour modelling. The first state to define is the starting state, *Idle*, when no actions haave taken place, see Figure 14. Then we follow the reply scenario, see Figure 7. The first stimulus that can be generated from the subscriber is *offHook*, resulting in the *DialTone* state. Next step is to enter a number and the subscriber state moves into *RingTone*. The called part (called B-part) answers the call and the state is moved into *Talking*. Finally when they close the call with *onHook*, the subscriber is back to the *Idle* state.

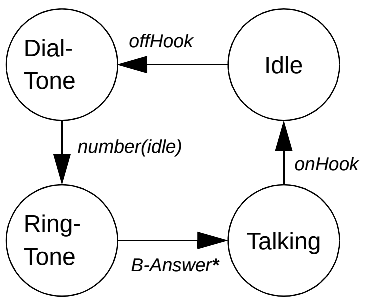


Figure 14. *Behaviour model for service NCC, scenario Reply.*

All the other scenarios are taken into account in the model, resulting in the model in Figure 15. It can be noted that there are a few new labels on the transitions in addition to the messages in Table 5. Timeouts are modelled as stimuli. There are also transitions labelled B-Answer and B-Calling which involve another behaviour model, denoted with asterisk in the figure. These are replaced with links, meaning that transitions in the other behaviour model causes a transition in the current model.

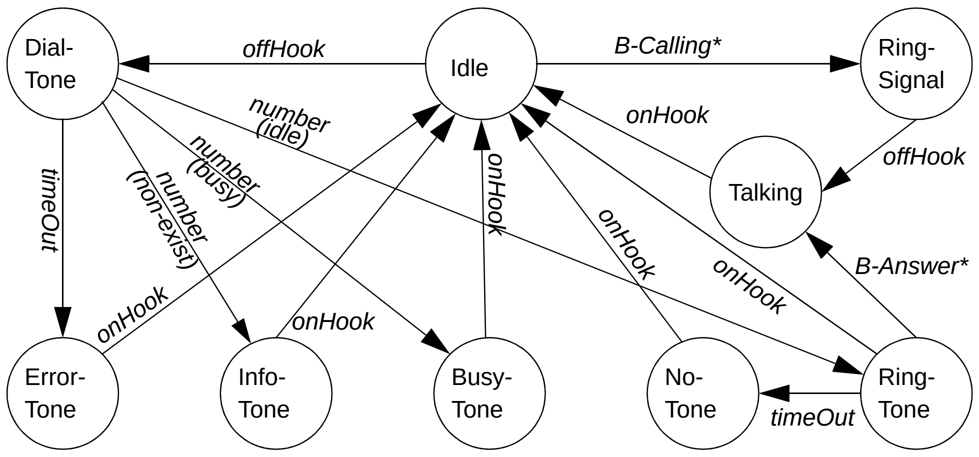


Figure 15. *Behaviour model for service NCC, all scenarios integrated.*

Define the behaviour usage profile. When the behaviour models are ready, the behaviour profile can be defined. There is no quantitative information on the system usage in the use cases, so this information has to be collected elsewhere. Typical information sources are measurements on earlier releases of the system and interviews with intended users of the system.

Two usage profiles are defined, for the NCC behaviour model, one for each user sub-type. Fictitious data is presented in the tables below.

Table 6. *Behaviour profile “Subscriber”*

State	Transition	**Subscr$_{lo}$**	**Subscr$_{hi}$**
Idle	offHook	1.0	1.0
DialTone	number(idle)	0.70	0.60
	number(busy)	0.25	0.30
	number(non-exist)	0.03	0.03
	timeOut	0.02	0.02
RingSignal	offHook	1.0	1.0
RingTone	onHook	0.98	0.98
	timeOut	0.02	0.02
Talking	onHook	1.0	1.0
BusyTone	onHook	1.0	1.0
NoTone	onHook	1.0	1.0
ErrorTone	onHook	1.0	1.0
InfoTone	onHook	1.0	1.0

The state weights represent the frequency of use when being in the respective states of the behaviour model.

Table 7. *State weights.*

State weight	**Subscr$_{lo}$**	**Subscr$_{hi}$**
W_{Idle}	1	2
$W_{DialTone}$	100	100
$W_{RingTone}$	100	100
$W_{RingSignal}$	50	50
$W_{Talking}$	15	10
$W_{BusyTone}$	100	100
W_{NoTone}	100	100
$W_{ErrorTone}$	100	100
$W_{InfoTone}$	100	100

The state weights for the idle state show that a subscr$_{hi}$ has twice as high frequency for starting a talk; the state weights for the talking state show that a subscr$_{hi}$ talks 50% longer than a subsc$_{lo}$.

Define the hierarchy usage profile. Finally the hierarchy profile is calculated, based on the state weights for each service.

$$P_i = \frac{W_i}{\sum_{k=1}^{16} W_k}$$

The concept translations discussed are, as shown in the example, not sufficient for automatic transformation. There is still a need for skill and intellectual work in the creation of the usage model.

It can be concluded that the use case model can be transformed into a usage model. The environment and structure levels contribute to the hierarchy model with a few additional modelling decisions. The behaviour model derivation is supported by the structure and event levels, while the profile information has to be collected from other sources.

5.2 Approach 2: Integration by Model Extension

Instead of creating a completely new model by transforming the use case model into a state hierarchy model, we can adopt the principal ideas behind statistical usage profiles and create an extended use case model, complemented with event statistics. If we can create well defined semantics for how test cases can be generated directly out of the use case model, we will save the effort of making two different models. The model extension approach is illustrated in Figure 16.

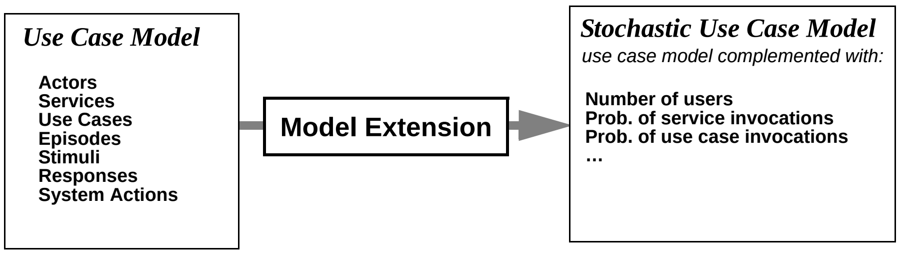

Use Case Model
Actors
Services
Use Cases
Episodes
Stimuli
Responses
System Actions
Model Extension
Stochastic Use Case Model
use case model complemented with:
Number of users
Prob. of service invocations
Prob. of use case invocations
...

Figure 16. *Extending a use case model with additional information needed for test case generation.*

The basic idea behind the model extension is to complement every part of the use case model where there are non-deterministic choices with the probabilities of the different choices.

Environment level. On the environment level, we have to extend the use case model with information on the number of instances of each actor and the probabilities for each actor to generate stimuli to the system. Furthermore, it has to be analysed if there are variants of actors with respect to their usage profile.

In our example, there are 7 instances of the subscriber actor and 1 instance of the operator actor. There are two variants of the subscribers, with respect to their usage frequency, subscr$_{hi}$ and subscr$_{lo}$. This information and the fictitous usage profile is summarized in Table 8.

Table 8. *Added information to the use case model on the environment level.*

	Actor		**Variants**		**Use Cases**
<0.95>	Subscriber (7)	<0.55>	Subscr$_{lo}$ (5)	<0.7>	Normal Call
				<0.05>	Activate CFU
				<0.05>	Deactivate CFU
				<0.2>	CFU Call
		<0.45>	Subscr$_{hi}$ (2)	<0.6>	Normal Call
				<0.05>	Activate CFU
				<0.05>	Deactivate CFU
				<0.3>	CFU Call
<0.05>	Operator (1)		none	<0.7>	Read Markings
				<0.3>	Reset Markings

Structure level. On the structure level we have to add profile information to the scenarios. To each branch in the HMSC flow (see Figure 4), a probability is attached. The resulting use case with profile information for the scenarios in the use case Normal Call with Charging is presented in Figure 17.

Event level. On the event level, probabilities are attached to each choice in the model. For example, the alternative operator introduces a non-deterministic choice between two or more alternatives. If we decorate the alternative operator, as shown in Figure 18, with probabilities, we can draw random numbers to decide which alternative is chosen during test case generation. This way we can construct stochastic semantics for each operator that determines how to generate scenarios. The scenarios are then used as test cases.

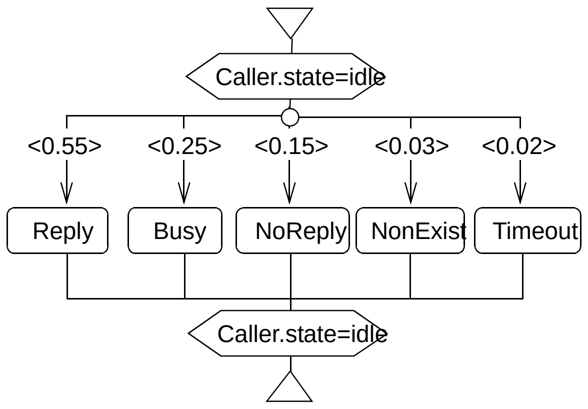
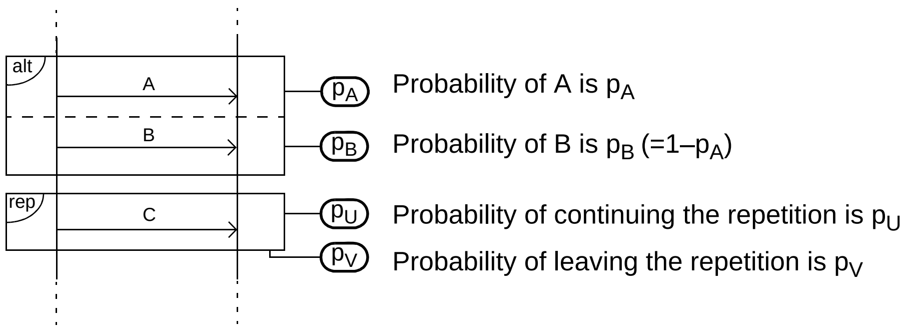


Figure 17. *MSC for the use case NCC extended with profile information.*



alt
A
B
rep
C
p_A
p_B
p_U
p_V
Probability of A is p_A
Probability of B is p_B (=1–p_A)
Probability of continuing the repetition is p_U
Probability of leaving the repetition is p_V

Figure 18. *Operators extended with probabilities.*

For each actor-service combination an *invocation probability* can be given to reflect how likely it is for this service to be selected for a given actor. As many users may interact simultaneously with the system, we need to specify the likelihood of the next event belonging to the same service invocation. To model this, we introduce for each possible actor-service combination a *continuation probability* that states the probability that the next event is within the same service invocation.

5.3 Discussion

Both the presented approaches to integration of use case modelling and statistical usage testing have shown to be feasible in a pilot study conducted on a PBX system, but there is a need for further investigation. This paper presents some examples from the pilot study, together with some preliminary observations and findings, but extensive case studies are needed to evaluate the two approaches, before deciding on which approach is preferable in which situation.

The *transformation* approach has the advantage of being based on two relatively mature disciplines which ends up in two models, each specifically defined for its purpose. The transformation rules support the modelling activities. The major disadvantage of this approach is the necessity of dealing with two different conceptual frameworks and, and having to perform the transformation between the models.

The *extension* approach has the advantage of not needing a second model, as it, instead, extends the modelling power of use cases with stochastic semantics. We can stick to the same conceptual framework for our requirements level usage model and decorate the model with probabilities of usage to enable reliability certification. Thus, the event based semantics does not need to be transformed into state based semantics.

6. Conclusions and Further Research

There remains many challenges in both requirements engineering and requirements-based system validation, and we believe that usage modelling will play an important role in both disciplines. Reliability certification is still in the cradle, but quantification of software quality will be a competition factor in the future, hence usage-based testing and a user perspective on the software are important. Use cases provide the means for communication between users and developers in the requirements phase, and usage-based testing allows for user evaluation prior to releasing the software.

The presented work addresses conceptual issues related to usage modelling and its application to both requirements engineering and testing. The objective is to integrate use case modelling and usage-based testing to form a comprehensive user-centred framework that enables both functional requirements specification and reliability certification. The presented results include a conceptual study of use case modelling and statistical usage testing based on the state hierarchy model. Both modelling techniques rely on similar concepts, which suggests that an integration is feasible. Two integration approaches are identified. The first approach aims at establishing transformations rules that allow use case models to be transformed into state hierarchy models. The second approach aims at extending use case models with stochastic semantics to allow test case generation directly from use case models. We believe that both approaches are feasible, but further research is needed to fully assess the virtues of each approach. Some of the areas where further research is needed are:

- Validation of rules for transformation of event-based use case models to state-based test models.
- Stochastic semantics of use case models for test case generation.
- Introduction of time in stochastic use case models.
- Empirical studies of an integrated usage modelling approach.

Acknowledgements. The authors would like to thank our colleague, Anders Wesslén for giving valuable comments on the paper. This work was financially supported by the National Board for Industrial and Technical Development, (NUTEK), Sweden, Reference P10505-1.

7. References

Booch, G., *Object-Oriented Analysis and Design with Applications*, Second Edition, Benjamin/Cummings Publ., 1994.

Bubenko, J. A., “Challenges in Requirements Engineering”, *Proceedings of Second International Symposium on Requirements Engineering*, pp. 160-164, York, UK, March 1995.

Buhr, R. J. A., Casselman, R. S., *Use Case Maps for Object-Oriented Systems*, Prentice Hall, 1996.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., *Object-Oriented Development - The Fusion Method*, Prentice Hall, 1994.

Dardenne, A., van Lamsweerde, A., Fickas, S., "Goal-directed Requirements Acquisition", *Science of Computer Programming*, Vol. 20, pp. 3-50, 1993.

Fowler, M., Scott, K., *UML Distilled: Applying the Standard Object Modelling Language*, Addison Wesley, 1997.

Graham, D. R., "Testing", *Software Engineering Encyclopedia*, (J. J. Marciniak, ed.), Vol. 2, pp. 1330–1353, John Wiley & Sons, 1994.

ITU-T *Recommendation Z.120*, Message Sequence Chart (MSC), International Telecommunication Union, 1996.

Jacobson, I., Christerson, M., Jonsson, P., Övergarrd, G., *Object-Oriented Software Engineering - A Use Case Driven Approach*, Addison-Wesley, 1992.

Jacobson, I., "A Growing Consensus on Use Cases", *Journal of Object-Oriented Programming*, pp. 15-19, March-April 1995.

Jarke, M., Pohl, K., Haumer, P., Weidenhaupt, K., Dubois, E., Heymans, P., Rolland, C., Ben Achour, C., Cauvet, C., Ralyté, J., Sutcliffe, A., Maiden, N. A. M., Minocha, S., "Scenario Use in European Software Organisations - Results from Site Visits and Questionnaires", Report of ESPRIT Project CREWS, no. 97-10, 1997.
Available via e-mail: crewsrep@informatik.rwt-aachen.de.

Karlsson, J., Wohlin, C., Regnell, B., "An Evaluation of Methods for Prioritizing Software Requirements", *Information And Software Technology*, (39) 14-15, pp. 939-947, 1998.

Kimbler, K., Wohlin, C., "A Statistical Approach to Feature Interaction", *In Proceedings of TINA'95*, pp. 219-230, Melbourne, Australia, March 1995.

Loucopoulos, P., Karakostas, V., *System Requirements Engineering*, McGraw-Hill, UK, 1995.

Mills, H. D., Dyer, M., Linger, R. C., "Cleanroom Software Engineering", *IEEE Software*, pp. 19-24, September 1987.

Musa, J. D., "Operational Profiles in Software Reliability Engineering", *IEEE Software*, pp. 14-32, March 1993.

Pohl, K., "The Three Dimensions of Requirements Engineering", *Proceedings of 5th International Conference on Advanced Information Systems Engineering*, pp. 275-292, Springer-Verlag, 1993.

Potts, C., Takahashi, K., Anton, A., "Inquiry-Based Requirements Analysis", *IEEE Software*, pp. 21-32, March 1994.

Regnell, B., Kimbler, K., Wesslén, A, “Improving the Use Case Driven Approach to Requirements Engineering”, *Proceedings of Second International Symposium on Requirements Engineering*, pp. 40-47, IEEE Computer Society Press, March, 1995.

Regnell, B., *Hierarchical Use Case Modelling for Requirements Engineering*, Technical Report 120, Dept. of Communication Systems, Lund University, Tech. Lic. dissertation, 1996.

Regnell, B., Andersson, M., Bergstrand, J., “A Hierarchical Use Case Model with Graphical Representation”, *Proceedings of International Symposium and Workshop on Engineering Computer-Based Systems*, pp. 270-277, IEEE Computer Society Press, March, 1996.

Rolland, C., Ben Achour, C., Cauvet, C., Ralyté, J., Sutcliffe, A., Maiden, N., Jarke, M., Haumer, P., Pohl, K., Dubois, E., Heymans, P., “A Proposal for a Scenario Classification Framework”, *Requirements Engineering Journal*, 3:1, 1998.

Rumbaugh, J., Blaha, M., Lorensen, W., Eddy, F., Premerlani, W., *Object-Oriented Modeling and Design*, Prentice Hall, 1991.

Rumbaugh, J.,“Getting Started - Using Use Cases to Capture Requirements”, *Journal of Object-Oriented Programming*, pp. 12-23, June 1994.

Runeson, P., Wohlin, C., “Usage Modelling: The Basis for Statistical Quality Control”, *Proceedings 10th Annual Software Reliability Symposium*, Denver, Colorado, pp. 77–84, 1992.

Runeson, P., Wohlin, C., “Statistical Usage Testing for Software Reliability Control”, *Informatica*, Vol. 19, No. 2, pp. 195-207, 1995.

Runeson, P., Wesslén, A., Brantestam, J., Sjöstedt, S., “Statistical Usage Testing using SDL”, *SDL´95 with MSC in CASE*, pp. 323-336, edited by R. Braek and A. Sarma, Elsevier Science B. V., 1995.

Runeson, P., Wohlin, C., “A Dynamic Usage Modelling Approach to Software Reliability Engineering”, In *Models for Estimation of Software Faults and Failures in Inspection and Test*, pp. 119–146, PhD thesis, Department of Communication Systems, Lund University, Lund, Sweden, 1998.

Selic, B., Gullekson, G., Ward, P. T., *Real-Time Object-Oriented Modeling*. Wiley & Sons, 1994.

Weidenhaupt, K., Pohl, K., Jarke, M., Haumer, P., “Scenario Usage in System Development: A Report on Current Practice”, *IEEE Software*, March, 1998.

Wesslén, A., Wohlin, C., “Modelling and Generation of Software Usage”, *Proceedings Fifth International Conference on Software Quality,* pp. 147-159, Austin, Texas, USA, 1995.

Whittaker, J. A., Thomason, M. G., “A Markov Chain Model for Statistical Software Testing”, *IEEE Transactions on Software Engineering*, Vol. 20, No. 10, pp. 812–824, 1994.

Wohlin, C., Runeson, P., “Certification of Software Components”, *IEEE Transactions on Software Engineering*, Vol. 20, No. 6, pp. 494-499, 1994.

Wohlin, C., Regnell, B., Wesslén, A., Cosmo, H., “User-Centred Software Engineering - A Comprehensive View of Software Development”, *Proceedings of Nordic Seminar on Dependable Computing Systems*, Denmark, August 1994.