Requirements Engineering in a Telecommunication Environment

Dirk Auchter, Johan Blom, Roland Bol
Uppsala University
P.O.Box 325
S-751 05 Uppsala, Sweden

Lars-Åke Fredlund
SICS
P.O.Box 1263
S-164 28 Kista, Sweden

Tomas Grelsson
Telia Engineering
Vitsandsgatan 9
S-123 86 Farsta, Sweden

Abstract

This paper presents the main result of the ARENA project: a method for requirements engineering in a telecommunication environment. The main characteristics of the method are a clear separation of general, specific, and formal requirements, the early construction of a moderately object-oriented model of a service, and an emphasis on traceability. The method has been tested on two case studies.

ASTEC
Advanced Software Technology

The Astec Requirements ENgineering Approach (ARENA) is part of the Astec project, supported by NUTEK, Telia Engineering, and Uppsala University. The authors wish to thank Anders Bengtsson, Mads Dam, Anders Ekman, Jesper Jonasson, and Bengt Jonsson for their contributions to the project. Please direct correspondence about this paper to Roland Bol (`rolandb@csd.uu.se`) or Tomas Grelsson (`Tomas.U.Grelsson@telia.se`). This report is available at `http://www.docs.uu.se/astec/Reports/9710.ps.gz`.

1 Introduction

This paper presents the main result of the ARENA project: a method for requirements engineering in a telecommunication environment. The main characteristics of the method are

- A clear separation of general, specific, and formal requirements.
- The early construction of a moderately object-oriented model of a service.
- An emphasis on traceability.

The method has been described in an earlier version of this paper [1]. It has now been tested on two case studies. These tests have been successful, but as expected some changes to the method have been made. Also our understanding of the requirements engineering process in the context of other aspects of service creation has been improved.

1.1 Motivation and History

Three partners are involved in the ARENA project: the two academic partners SICS and Uppsala University, and the industrial telecom partner Telia. The motivation for Telia in participating in this project is that telecommunication companies today provide a diverse service portfolio containing more and more complex services. Hence, to satisfy demands for shorter development cycles, telecom companies today must be able to produce and handle high-quality requirements specifications by means of a more systematic product development [8, 9].

The main motivation for SICS and Uppsala University is to gain experience from developing and applying formal specification and analysis methods in the context of real life examples. In particular, we wanted to test a method for the automatic detection of feature interactions [3]. When the project started, it turned out quickly that the necessary formal descriptions of features[1] on the requirements level did not yet exist. Formal descriptions, if present, describe features at a much lower level of abstraction than our analysis method is intended for. The level of abstraction of our method corresponds to descriptions produced during requirements engineering; yet these descriptions are informal and often very general. This was felt as a shortcoming, not just for the purpose of interaction detection, but also in the quality of the requirements specification itself.

Therefore we decided to start from the beginning, and propose a structure of the requirements engineering process that separates general requirements from specific ones, that ensures that the specific ones are indeed written down, and thereby become available for formalisation. We also studied which formalisms would be most suitable from the point of view of requirement engineering (as

[1]We use the words "service" and "feature" almost interchangeably in this paper. Often a service can be considered as a package of smaller features, but a clear boundary does not exist.

opposed to automatic interaction detection and analysis). This part of the project is reported in another paper [7].

We obviously needed a real case study to learn the current practices and test improvements. To this end we used the requirements specification of a Voice Mail service [14]. Later, we followed the development of the requirements specification of another service, a Virtual Private Network. Apart from borrowing some simple examples, this paper does not report on these case studies.

1.2 Tool support

Working with a real case study meant that we were faced with the real size of a requirements specification. Handling more than a toy example requires support from a software package. Moreover, its complex structure, in particular the (traceability) links between items, can hardly be represented on paper. Therefore we identified already at the start of the project the need for a Requirements Management System. Essentially, this system should be a database containing items and links, attributes and a history. It must be able to make selections of items based on their links and attributes, and present these in a user-friendly way. It should be able to link formal requirements and results of formal analysis by cooperation with tools supporting the formalism.

We have used the tools DOORS version 2.1 [6] and RTM version 3.6 [13]. In short, our conclusions are that DOORS can be characterised as a light weight tool: easy to learn, but the performance and user interface of this version did not handle large specifications very well. We found that RTM was more suitable to our needs in that respect, but its capabilities come at a price: RTM is a much larger tool, based on an Oracle database, so it requires more support and training. A complete evaluation of these tools and their use in supporting our method can be found in [2].

1.3 Results

Our current view of the requirements engineering process is summarised in Figure 1. The boxes represent collections of requirements and/or a model of the service. The irregular blobs represent activities. The detailed explanation of this figure comprises most of this paper. It can be divided into three parts.

The first part, actors, general requirements, and a core concept of a new service, exist at the beginning of the requirements engineering process. *Actors* are the people and organisations (both outside and inside the service provider), that have an interest in the well-functioning of the service. On a very general level, many of their requirements, such as legislation, external and internal standards, are the same regardless of the details of the service. These requirements we call *general requirements*. It is usually one of the actors who comes up with the idea initiating the process: the *core concept*. These inputs are discussed in Section 2. There we also discuss the absence of the network and existing services in Figure 1.

In Section 3 we discuss the second part of the figure: the process that leads from the inputs to a conceptual model and detailed requirements for a particular

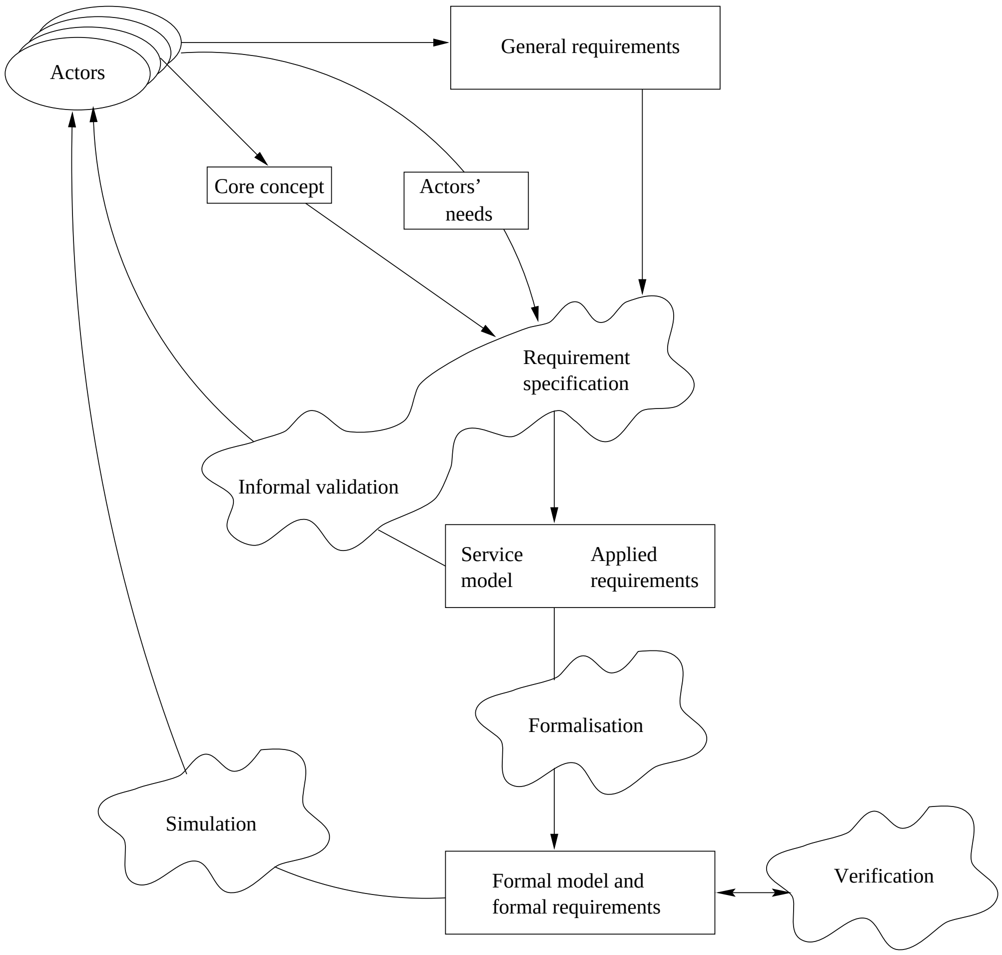

Figure 1: Picture of the requirements engineering process

service. Based on the core concept, all involved actors formulate their view on the new service: the positive effects they expect to obtain from the service, and the conditions the service should satisfy. The result we call *actors' needs*. The core concept will be close to the needs of the initiating actor.

The requirements specification process then consists of collecting actors' needs, interpreting general requirements (which ones are applicable and what do they require of this particular service), organising the material, and resolving conflicts between requirements. This process results in the applied requirements and a detailed model of the service. This *service model* contains more functionalities (that may follow from general requirements, for example error handling and customer support), and more detail than the core concept. Actors' needs and applied requirements grow as the service model develops, and more general requirements become applicable to the current model of the service. During this stage there is a constant validation of the produced model and requirements w.r.t. the intentions of the various actors. We call this *informal validation* to distinguish it from validation of a formalised model and requirements.

Formalising a suitable subset of the requirements, formal validation (for example by simulation) and formal verification form the third part of Figure 1. Some remarks about formalisation and its aims are made in Section 4.1. The discussion there remains brief; for a more concrete discussion we refer to a satellite paper [7].

After the conclusions in Section 5, Section 6 summarises the terminology used in the paper.

2 Inputs of the Requirements Engineering Process

In this section we discuss the inputs of the requirements engineering process. A new service is added to an existing telecommunication system, and in an existing organisation. This organisation provides the background for a new service: actors that are involved and the requirements they have about services in general. When the requirements engineering process is started for a service, there is also an existing idea (probably limited and perhaps vague) of what the service should accomplish.

2.1 Actors

Actors are those entities that perform activities in the telecommunication system, and in particular in the new service. Some actors are customers of the service provider (most notably subscribers and users), but many actors are branches of the service provider that have a specific function related to the service (customer support, billing, marketing, ...).

For different activities we have different actors, but two or more actors can be combined into a single person or organisation. For example, we will differentiate between subscribers (who sign contracts, pays bills, etc.) and users (who make calls, use features, etc.). These are different actors, that are often combined in a single person. A person or organisation combining different actors is called *stakeholder*.

W.r.t. a specific service, an actor can have one or more *roles*. For example, a user can be either caller or callee in an ordinary call. Using a simple feature like call-waiting, the user can be engaged in several calls at the same time, being the caller in one and the callee in another.

The set of actors will remain almost constant; only occasionally a new service will introduce a new actor. Therefore we can consider the set of actors fixed during the development of a service. In contrast, which roles exist depends on the service.

2.2 General Requirements

General requirements are not specific to any service, but can apply to more than one, perhaps many, services. An important part of the requirements engineering process is to identify which general requirements are mandatory for the new service, which ones are optional, and which ones are not at all applicable.

Like the set of actors, the collection of general requirements is stable (relative to the rest of the process). We therefore consider it as input. We assume that each actor has compiled a list of its general requirements. For Telia, this assumption is not yet realistic, but different categories of general requirements have been identified, and a sizable collection has been built. These requirements come from legislation, standards (national and international), and from inside Telia.

Some examples of general requirements are:

> "The service must comply with the interfaces for other services."
> "The customer must be able to order the service through the existing ordering system."
> "The service must make a profit."
> "The service must provide relevant statistics about its behaviour."
> "The service may be unavailable at most n hours per year."

2.3 The Core Concept

Each new service starts with a central idea. We propose to capture this idea into a so-called *core concept*. It describes the functionalities that are central for the service, and how it is intended to be used. Usually it does not describe optional extras, handling of errors, ways to (un)subscribe, available user support, etc.

What is included in the core concept will depend heavily on which actor the idea comes from. For example, an idea can arise from a new technical possibility (e.g., a large memory that can contain millions of messages), but also from the fact that a competitor offers a certain service (e.g., voice mail). In the second case, the emphasis of the core concept would lie on the functionalities that must be offered, and perhaps even mention prices. In the first case, the core concept might name only a few of the functionalities that are finally included, but be more specific about resources. The core concept is usually incorporated later in the actors' needs document of the initiating actor.

In most cases, the requirements engineer is not consulted immediately after the idea arises. Instead, the idea is validated by a preliminary study. Depending on the character and origin of the idea, this study can be for example a feasibility study or a market analysis. The results of this study are incorporated in the core concept.

2.4 Other Services and the Network

The network and other services exist during the development of a service. The network will not change much; other services may change but mostly independently of the creation of the current service. In this sense, the network and other services should also be considered as input.

Nevertheless, we have chosen not to represent them explicitly in the information diagram that we present in the next section. The qualitative constraints that the network places on the service are represented as general requirements - mostly standards. Quantitative constraints (capacity, rate of failure) are usually propagated in the other direction: the requirements on the service, coming from the actors, must be met by a design based on the limitations of the network. A feasibility study can prove that such a design exist, or show how the network should be improved, but otherwise these considerations are not part of the requirements capture.

Practical experience shows that the creation of a service is difficult enough in isolation, and that unplanned interactions with other services are not considered. Indeed, Eurescom Project P509 [11] recommends that a Service Interaction Handling Process is created separate from the Service Creation Process. One of the goals of the method in this paper is to create a Service Creation Process that provides useful input to the Service Interaction Handling Process.

3 The Information Diagram

The requirements engineering process builds a structure of modules containing items of information. Such an item can for example be a requirement, or an object in a model of the service. Items can have many attributes, such as date-of-creation, priority, etc. Figure 2 shows the top level of this structure, which we call the information diagram.

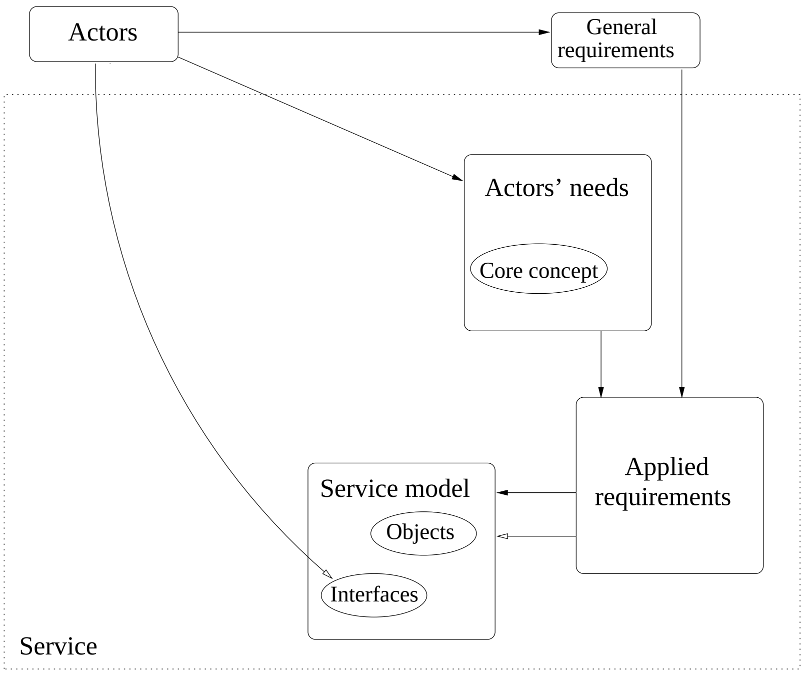

Figure 2: Picture of the information diagram

This figure is quite similar to Figure 1; the main difference is that Figure 1 is dynamic: it shows the process, whereas this figure is static: it shows how information is structured. The process will be recorded in the attributes of items.

The modules "actors" and "general requirements" are not service-specific; the other modules are. Arrows with filled tips denote sets of *traceability* links. A traceability link shows the origin of an item, for example the general requirement that gave rise to a specific requirement. The other two arrows are explained in Section 3.1.2 and 3.2 respectively.

3.1 A Service Model

In order to write concise requirements, we should have something to write *about*. That is, we need a model of the service. At the start, this model will be rudimentary, but later it will become more developed. These models serve as a frame of reference for the requirements.

We call such a model a *service model*. A service model is a *logical* model of a system or service, which should be independent of design structure and implementation.

3.1.1 Objects, Attributes, and Operations

A service model typically describes a set of *object classes*. The service creates (on a very abstract level) one or more instances of these classes: *object instances*. For example, a voice mail service might create and delete instances of the object class **message**. In the rest of this paper, we will only deal with object classes, not with their instances. Abusing terminology, we will use the word "object" for "object class".

Objects are not black boxes: they have a visible structure of *attributes*. For example, the object **message** could have attributes like **size** and **timestamp**. These attributes are part of the service model, and should not be confused with the attributes that the requirements engineering process assigns to items in the information diagram, stating for example when the introduction of **timestamp** was approved.

Furthermore objects provide *operations* to their environment. Objects are related statically (an object having as an attribute a reference to another object) and dynamically (an object using an operation from another object).

The creation of a service model from a set of requirements in natural language means to change the representation of certain requirements: some requirements will become part of the service model. In other words, if a requirement states that an object, attribute, or operation exists, then the requirement is represented by the presence of this object, attribute, or operation in the model. For example, the requirement that the contents of a message are stored can be represented by the existence of an attribute **contents** of the object **message**. Traceability of requirements makes it necessary to annotate which requirement(s) gave rise to the introduction of the object (attribute, operation). This is the purpose of the traceability links in Figure 2. Section 3.4 gives a detailed account of these relations.

3.1.2 Interfaces

The operations in the service model are organised as belonging to objects. But we also need to maintain an orthogonal view: which operations are available for a given actor, and how they can be combined. We define an *interface* as the combined functionality that operations (from one or more objects) provide to an actor. This explains the link between an actor and his interface.

Just like attributes may become more and more complex, and finally become objects, operations may become more and more complex and evolve into (parts of) interfaces. For example, "getting help" could evolve from a single operation to a set of operations (which one of them is appropriate might depend on the user, the phase of the call, the time of the day, and other parameters).

Interfaces can also be combined, e.g., a user can at first perhaps only use the normal POTS operations and an additional services subscription operation. After subscribing to a service, new operations become available in the user interface. This gives rise to new dialogues.

3.2 Requirements

With the service model, the requirements engineering process must produce a set of requirements that are sufficiently specific for further use, during design or formalisation. Many of these requirements arise from the application of a general requirement to this particular service. Therefore we call these *applied requirements*, although there are other sources of applied requirements as well (as we point out in the next section).

Applied requirements are statements about objects, attributes, and operations in the service model, not requiring their existence, but telling more about them. For example, the requirement can say that a message is available at least 48 hours after recording. Such a requirement must be linked to the object (attribute, operation) it refers to, but this link is not a traceability link; we call it a *conformance* link.

Requirements regarding user interaction often cannot be linked to an individual operation, but refer to an interface as a whole. A particular form of such a requirement is a *dialogue*, which specifies a possible sequence of interactions between an actor and various operations from his interface. (Sometimes more than one actor is involved. For example, when a user asks the system to connect him to an operator.)

Experience has shown the need for an intermediate module called *actors' needs*. This module contains "new" requirements that are prompted by the current service description (as opposed to instances of general requirements). The step from actors' needs to applied requirements involves joining different formulations of the same requirement, ranking requirements according to priority, and perhaps even discarding requirements that are inconsistent with other ones, or simply too expensive for their importance.

3.3 Evolution

The aim of the requirement engineering process is to derive a useful service model and applied requirements from the core concept and general requirements. During this process, there will be constant contacts with the actors that are responsible for the general requirements. In this way the applied requirements are *validated* w.r.t. the actors.

Initially, the service model consists of some objects and operations that represent basic notions in the core concept. We sometimes call this model the *core model*. This model is subsequently extended to the full service model, the *extended model*. Our experience shows however that a sharp distinction between core model and extended model is not useful. For example, the importance of an object is not necessarily related to it being in the core model; an attribute reflecting importance is easier to use and more accurate.

The extended model and applied requirements can arise from the core model and general requirements in several ways.

- The application of a general requirement to the new service.
  This will involve several steps. For example, consider the requirement "The service must provide relevant statistics about its behaviour." This requirement is obviously not specific enough for any use. The requirements specification group should consult the source of the general requirement (i.e., the actor), and 'negotiate' a concrete set of "relevant statistics" (objects, attributes) and how they are to be "provided" (operations, the interface, dialogues). At first, the core model will serve as the frame of reference for these negotiations, but extensions of the core model can introduce additional "relevant statistics." In the end, the extended model itself is the frame of reference.

- The addition of functionality.
  For example: error handling, help functions, billing, (un)subscription. In short, all operations that are not contained in the core model, but that are necessary in a complete service. Usually there will be general requirements that require the existence of these functionalities, and constrain their behaviour. But it also happens that actors invent new functionality because they feel a need for it; naturally then this functionality is first incorporated in the actors' needs, and its feasibility evaluated, before being added to the applied requirements and the extended model.

- Concretisation.
  This involves defining in a more precise way how a certain functionality is achieved.

The key issues are that the service model and applied requirements are developed together, and that the actors decide how their general requirements must be interpreted. It is necessary that conformance links are maintained such that one can easily obtain all requirements about a particular object or part of it. And it is necessary that traceability links are maintained in order to know which actor(s) must be consulted on a particular requirement. Other reasons for maintaining traceability links are discussed in the next section.

3.4 Traceability

During specification and development of services and systems there is always a need to understand how requirements and models are related. Typical questions are:

- Have all requirements been realised in the set of requirements on lower levels?
- What is the background of a certain requirement (for example to judge its importance or to verify if this background has not changed)?
- What is the impact of a proposed change?

In order to be able to analyse in an effective way if a product satisfies all its requirements, these have to be traceable through all levels of as well the requirements as the realisations. With the term traceability is understood that the relations from the requirements to each other, to design, and to test cases in some way are made visible in the models of systems and requirements. Thus traceability extends well beyond the requirements engineering phase of a project. On the other hand, if traceability is not maintained during requirements engineering, attempts to obtain good traceability later will be futile.

Figure 2 concentrates on *vertical* traceability during the requirements engineering process: the possibility to relate the evolution of requirements in different phases. The direction of the arrow serves only to show a direction of evolution: traceability must be supported in both directions. Traceability also requires a complete history of the development of all data in a fixed location: which changes were made, when, why, and by whom (actor and physical person). The figure does not show this *horizontal* traceability.

Traceability links are in general many-to-many relations: a general requirement may be broken down into several applied requirements; at the same time an applied requirement may contribute to fulfilling several general requirements.

Some aspects of traceability are not shown in the figure. For example, for each general requirement, we need not only to know

- which actor(s) generated it (this classification is static, i.e. service-independent),
- if it is considered applicable to the service (this classification is dynamic, i.e. service-dependent),
- to which applied requirements it led,

but also

- if these applied requirements realise the general requirement completely,
- if it is (in)consistent with other general requirements,
- its place in other classifications, both static and dynamic.

4 Goals of the Requirements Engineering Process

In this section we study the goals of the requirements engineering process: who are the consumers of the information produced? We begin with a discussion on formalisation. Formalisation should be part of the requirements engineering process, but it is currently hardly there. We feel that one of the causes for this omission is that the formalisation process lacks proper input: when the step from general requirements to applied requirements is not made, one ends up trying to formalise general requirements, which usually turns out to be impossible.

The most obvious goal of the requirements engineering process is to provide input to the design of a service, eventually leading to its implementation. The requirements themselves give rise to test cases, which are used to verify that the requirement is met. Finally, as the Eurescom Project P509 [11] recommends, a separate Service Interaction Handling Process should take care of service interactions. This process must receive its input from the Service Creation Process in an early stage of the service creation, because its output must be considered during design and implementation. Here in particular a formalisation of the service could provide great benefits, as there are many studies on how to detect feature interactions based on their formal specifications [4, 5].

4.1 Formalisation

Once a requirement or a part of the model is sufficiently specific, it might be suitable for formalisation, i.e., it might be formulated in a formal language. Generally speaking, formalisation has two purposes. Firstly, it serves to exclude ambiguities that arise from natural language. Secondly, it is the starting point for further development (design), validation (e.g., in the form of simulation), and verification. The formalisation provides a description of (parts of) the service that is understandable by a computer, which is a necessary prerequisite for automating these processes.

Our opinions on what parts of a service are most apt to be formalised, what formalisms are suitable, and which analysis techniques can be used, can be found in a separate paper [7]. It is noted there that *use cases* play a central role. (Use cases resemble dialogues, but the communication between different objects is not hidden.) Here we concentrate on the integration of formal methods in the requirements engineering process.

The formalised requirements should logically be part of the whole requirements specification. That is, Figure 2 should be extended with two more modules, containing a formal model and formal requirements. Traceability should be maintained from these modules back to the service model and applied requirements. Conformance links between the formal model and requirements should correspond to conformance links between the service model and applied requirements. Formal and informal descriptions are complementary for building understanding, but in case of ambiguity or contradiction, the formal requirements must be followed.

After formalisation, we can apply a multitude of analysis techniques to answer important questions like "Does the formal model satisfy the requirements?" and "Are the requirements consistent?". This process is called *formal verification*.

The formalisation can also be used to present automatically generated dialogues to actors, in other words, one can build a simulator of the service. In this way one can validate that the requirements as stated by the actor correspond to his actual desires.

Formal verification and validation provide an attractive early check on the correctness of the requirements specification. Correcting mistakes in this stage is much cheaper than correcting them later during design and implementation.

4.2 Design

During design, one builds again a model of the service, but this time one that is based on the actual architecture of the system. In telecommunications, this architecture is mostly given, and large parts of it are relatively old. As a result, the design may deviate quite much from the service model built during requirements engineering. In this situation, extra attention must be paid to maintain traceability - practice is often the opposite: no traceability information is recorded.

During design and implementation, it may turn out that certain decisions that were made before lead to high costs, and must be changed. But a balanced judgement about such a change cannot be made if the reasons for the original decision cannot be traced. This is one of the main reasons for maintaining traceability information.

The transition from requirements to design is a topic for further study of the presented method.

4.3 Testing

The information diagram can be extended with a module recording test cases and test results, with the appropriate traceability information. This should make it simpler to decide that a requirement is sufficiently tested. Testing is formal and automated, so it is to be expected that most tests can be linked to formal requirements. In particular, there is a natural correspondence between use cases and executions of the system. We have not studied current testing practices yet in our project.

4.4 Service Interaction

Any process for the detection of service interaction needs descriptions of the services as input. Preferably, these descriptions are uniform (in the same format), if possible formal. The preparation of such a description is part of the requirements engineering process. The advantage of the method described in this paper is that such descriptions are an *integral* part of it; they are not just produced, but also used to improve the requirements specification. We plan to study the relationship between service creation and interaction detection further, but this depends on the installment of an interaction handling process such as the Service Interaction Handling Process proposed in [11].

5 Conclusions

With some small modifications that are now incorporated in it, the method has proved to be workable. Yet what is lacking is an evaluation of its output. Our first case study was in fact an exercise in re-engineering: we used a requirements specification from a project that was already finished. In the second case study, we managed to follow the development of a requirements specification in the traditional way, but time to compare our structure with the other one was lacking. So it is too early for a definite conclusion.

From Section 4, it is clear that our work is not finished. Apart from the application of the method in an actual development project, the method needs to be extended towards the processes that it serves: formalisation, design, and interaction analysis. Regarding formalisation, the first results are already obtained [7].

The extension towards design requires a choice of architecture. The IN hierarchy [10] appears to be the most interesting one, and well suited to continue the application of the method: the planes of the IN hierarchy could be added as different modules to the Information Diagram. A formalisation of levels in the IN hierarchy would allow a deeper analysis than possible on the requirements level.

An extension of the method towards a service interaction handling process is the most exiting possibility. It combines the origins of the ARENA project, formalisation and interaction analysis, with its strong points of structure, traceability, and separation of concerns. As this process does not exist yet, the ARENA method could be an integral part of its organisation, rather than requiring a change of routines.

6 Terminology

We conclude with a summary of the terminology introduced. The briefness of these descriptions makes them less exact than the explanations given elsewhere. We have not tried to eliminate circularity: these are not mathematical definitions. There are many such lists of object oriented terminology, e.g., [12, 15], which unfortunately differ. This list is no exception.

Actor A function that a stakeholder has w.r.t. services. Example: "subscriber" and "user" are two actors, which can be the same customer (in case of your home phone) or different ones (in case of your work phone). Actors are service independent (with perhaps a few exceptions).

Applied requirement A requirement that is specific to the service. It is usually an instance of a general requirement in which the "variable notions" are made concrete. An applied requirement can be linked to an object, attribute, operation, or interface in the model.

Attribute Part of an object that contains information. It can be a reference to another object. For example, roles are usually attributes that refer to an actor.

Core concept The central and initial idea for the service as written down.

Core model The service model that represents the initial basic idea, and therefore contains only those objects, attributes and operations that are essential for this idea.

Dialogue A sequence of interactions between an actor (sometimes more than one) and the system (interface). In other words: a use-case as observed by the actor: the system is considered to be a black box.

Extended model A service model that captures the full functionality of the system.

Formal model A model defined in a formal language, with a formal semantics.

Formalised requirement A requirement stated in a formal language about a formalised part of the model. Example: a Message Sequence Chart can formalise the requirement that a certain use-case or dialogue is possible.

General requirement A requirement that is stated such that it can reasonably apply to several services.

Horizontal Traceability The possibility to retrieve and relate different instances of requirements (e.g. in time, or as formulated by different actors) in a single location in the information diagram.

Interface The combined functionality that operations from various objects provide to an actor. In other words, the functionality of the model of the system as perceived by an actor.

Object (class) A coherent part of the model that is complicated or important enough to be modeled as a separate entity. The judgement involved in this definition rests with the requirements engineer.

Object instance A service may involve one or more instance of an object class. Standard cases are one static instance (for example a global clock), and many instances that are created and deleted (for example messages).

Operation Part of an object that provides a functionality to the environment. Typical examples: informing the environment about a part of the object; allowing the environment to modify a part of the object.

Role The role of an actor in relation to an object. Example: a user can be the sender or the receiver of a message. Roles are service dependent, although similar services will involve similar roles.

Service model Collection of objects, with their attributes and operations, to which requirements can refer.

Simulation A form of validation, that uses a formal description of the model (and possibly requirements) to present the actor with a dialogue.

Stakeholder An entity (human, organisation, or system) involved in using, providing, and/or managing services. Example: "customer".

Use-case A sequence of interactions (trace) that is possible between two or more objects in the model. Objects interact through their operations.

Validation The activity of checking a part of the model or requirements against the desires of an actor.

Verification The activity of checking the internal consistency of the formal model and formalised requirements.

Vertical Traceability The possibility to relate requirements through different levels of abstraction (general requirement, applied requirement, formal requirement, design, implementation).

References

[1] ARENA Project. Requirements Engineering and Formalization in a Telecommunication Environment. Technical Report 96/02, ASTEC, March 1996.

[2] D. Auchter. Tool Support for Requirements Engineering: Applying the ARENA Methodology. Master's thesis, Uppsala University, April 1997. Technical Report 97/05, ASTEC.

[3] J. Blom, R. Bol, and L. Kempe. Automatic detection of feature interactions in temporal logic. In K.E. Cheng and T. Ohta, editors, *Feature Interactions in Telecommunications Systems III*. IOS Press, Amsterdam, Netherlands, October 1995.

[4] K.E. Cheng and T. Ohta, editors. *Feature Interactions in Telecommunications Systems III*. IOS Press, Amsterdam, Netherlands, October 1995.

[5] P. Dini, R. Boutaba, and L. Logrippo, editors. *Feature Interactions in Telecommunications Systems IV*. IOS Press, Amsterdam, Netherlands, June 1997.

[6] DOORS PC Reference Manual, version 2.1. Oxford, United Kingdom, August 1995.

[7] L.-Å. Fredlund. Formalising Telecommunication Services using the ARENA method. Technical report, ASTEC, 1997. To appear.

[8] M. Hjelte, B. Kranqvist, U. Nyman, and S. Oldgren. Krav på krav. Sveriges verkstadsindustrier, 1995. ISSN 1103-7008, in Swedish.

[9] IEEE Recommended Practice for Software Requirements Specifications. IEEE Computer Society, 1993. IEEE Std 830-1993.

[10] ITU-T. *Recommendation I.312/Q.1201*. October 1992.

[11] Kristofer Kimbler. Addressing the interaction problem at the enterprise level. In *Feature Interactions in Telecommunications Systems IV*, pages 13–22, June 1997.

[12] RACE Project 2049. Open services architectural framework for integrated services. RACE, March 1995.

[13] RTM User Guide, version 3.6. Camberly, Surrey, United Kingdom, September 1996.

[14] Telia Research. Telebud's requirements specification, 1994. Telia Internal Information, not externally available.

[15] TINA-C Glossary of Terms, Version 1.0 (Draft). TINA Consortium, 1996.