# From Requirements Capture to Design: Combining the ARENA and SOMT method

Dirk Auchter
dirk@csd.uu.se

November 25, 1997

**Abstract**

This paper makes an attempt to combine a method for requirements engineering with a method that integrates object-oriented analysis and (SDL-) design. For given examples of a Voice Mail service, requirements are distributed on an existing architecture.

## 1 Introduction

This paper presents experiments made combining a method for requirements engineering with a method that integrates object-oriented analysis and (SDL-) design. The motivation of these efforts was to prepare for a smooth transition from requirements capture to design. [1] presents a more developed method based on use case modelling. The two methods here, ARENA (Astec Requirements ENgineering Approach) [2] and SOMT (SDL-oriented Object Modelling Technique) [3] shortly presented in the next chapter, both aim for an improved service development process with the emphasis on requirements analysis and design.

Another issue of this paper is how requirements on a telecommunication service (here: a Voice Mail service) can be distributed on an existing architecture (here: the *Intelligent Network* [4] architecture). In [5] we demanded for a treatment of requirements at a lower level of abstraction in future works, as requirements therein were described rather informally and very general. In this paper we try to meet the demand for an early formalisation of requirements, which was already suggested in [6].

## 2 Outline for the method combination

### 2.1 The ARENA method

ARENA is originally proposed as a method for requirements engineering in a telecommunication environment. The method has been tested on two case studies. The main results of the ARENA project are documented in [2]. Details of the method can be found there.

The main characteristics of the ARENA 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 ARENA requirements engineering process documents its results in the following models:

- *actors* that have a potential interest in interacting with the service to be developed.
- *actors' needs*, a collection of requirements in terms of its respective actor(s), expressing how the new service should look like or behave. Starting-point is the *core concept*, the central idea behind the service.
- the almost stable set of *general requirements*, i.e. requirements that are not specific to any service, but apply to more than one, perhaps many services.
- *applied requirements* which are extracted from the set of general requirements and the newly compiled actor's needs, i.e. a set of more detailed (applied) requirements on the service.
- *service model*, an object-oriented logical model of the service including rudimentary (descriptions of) objects and interfaces.

A main goal of ARENA is to ensure traceability (both vertical and horizontal) between related requirements and between requirements and (parts of) the service model.

### 2.2 The SOMT method

In this section we give a very brief overview of the SOMT method. For details the reader is referred to [3]. Basically SOMT combines object-oriented analysis (using OMT [7] notation) with SDL-based [8] design. The development process according to SOMT is divided into five major activities the results of which are documented in different models. The following activities are distinguished:

**Requirements Analysis** The purpose of this first activity is to get an understanding of the problem domain. The system is viewed as a black-box and only objects and concepts visible on the system boundary and outside the system are modeled. Among other models as e.g. a data dictionary, this activity uses a *requirements object model*, an object model in its conventional form with diagrams showing objects and their relations, and a *use case model* that consists of a set of use cases described by structured text and/or MSCs [9].

**System Analysis** In order to achieve the required functionality, this activity identifies the architecture of the system and the most important objects to be represented in the system. The main models used in this phase are an *analysis object model* to describe the architecture and represent the objects, and an *analysis use case model* to describe the interactions between (some of) the objects.

**System Design** In this phase the system architecture and the interfaces between its parts are precisely defined. In this way the implementation structure and overall design strategies are identified. The architecture definition will usually stated by SDL system/block diagrams and the interfaces between the different components in the system by MSC diagrams.

**Object Design** In object design the functionality of the system (including the behaviour of all objects) is defined in detail. Most commonly this is done by use of SDL process graphs.

**Implementation** The purpose of this phase is to produce the (executing) application, implementing all requirements.

To ensure traceability between the models and to provide for a smooth and well-defined transition between them, SOMT introduces *implinks* and the concept of *Paste As*. An implink is a "directed relation between two objects, usually (but not necessarily) in different models". The *Paste As* concept is thought to support the task of creating implinks, "an object in one model can be copied and then *pasted as* a new object in another model".

### 2.3 The service development process

It is promising to combine a method that aims for re-use, validation and simple verification of requirements while ensuring traceability between its models (ARENA) with a method providing for simulation and enabling traceability between object models and SDL diagrams and/or MSCs (SOMT). Above all the preservation of traceability in a transition between the methods improves the service development process. A verification that all analysis concepts are actually implemented is made possible. Furthermore the effects can be checked if one analysis object is changed because of changed requirements.

As ARENA somewhat ends where SOMT starts, it seems natural to combine these two methods, i.e. to bridge requirements capture and analysis with the design of a service. It is not necessary (not even possible !) to define exactly the borderline between the methods but the transition happens somewhere during the construction of a (moderately) object-oriented model of a service. Figure 1 shows how responsibilities would be shared among the two methods in service development. The figure leaves out later stages in service development such as the implementation of the service.

Note that the arrows with filled tips denote the transition between different phases of service development. This vertical traceability is provided and implemented (at least for the SOMT part) by the *PasteAs* concept described above. For phase-internal links (horizontal traceability) tool support is required. Desirable would be one tool maintaining all links, both horizontal and vertical, or in case of more tools involved, an interface (interfaces) between them. For example provide both RTM[1] and SDT[2] a concept for linking objects (RTM

[1] A requirements management system.

[2] A toolset for SDL, OMT and SDL specifications.

having the more elaborated one). In order to preserve traceability even while switching between the tools (forwards and backwards) an interface is needed.

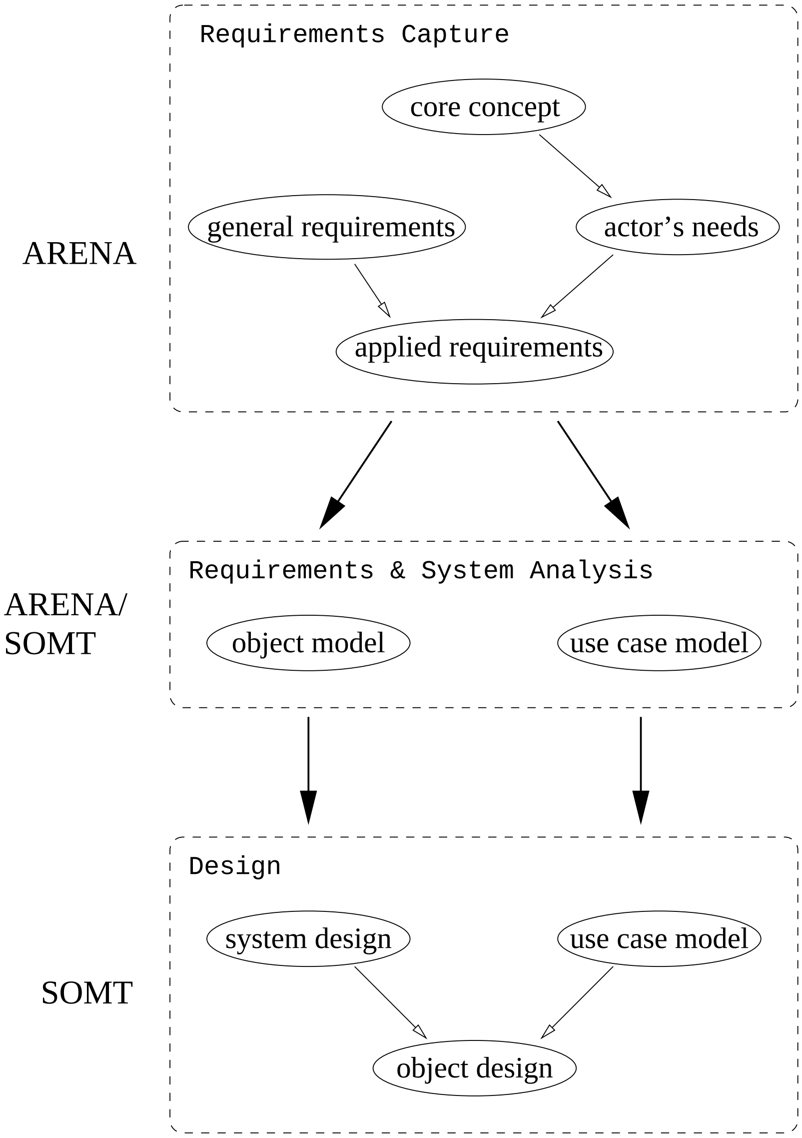


Figure 1: The service development process

## 3 Formalising Voice Mail requirements

As an attempt in formalising requirements we have specified a number of requirements on the Voice Mail service with the help of the tool SDT[3]. The requirements have mostly been derived from preexisting use cases in the original RBL-specification [10]. The work done in [5] served as input for our efforts. Requirements have been structured and linked according to ARENA and with help of a requirements management system. In the following we therefore assume the requirements capture phase to be completed.

### 3.1 Formalisation in requirements analysis

In this phase less effort was put in the construction of an (initial) object model (in ARENA terminology the *core model*). We were more interested in modeling use cases with MSCs as these were poorly covered in [5]. Therefore we described the most interesting functional requirements by a number of MSCs. These MSCs illustrate the interaction between the system (here: TeleBud) and the environment (here: User) and thereby formalize to some extent the functional requirements. The reason for abstracting away from the internal structure of the system (the Voice Mail service TeleBud) is to obtain more readable requirements upon the service interface. As an example see Figure 2 and 3. The examples illustrate that a user can be either transferred automatically to TeleBud on receiving a busy tone or in case of no answer (Fig. 3), or that a user can directly access TeleBud (Fig. 2). Note the final condition in the *DirectAccess* MSC. After the user has given the number of the recipient (`bnr`) and timepoints for delivery, he ends in the initial state of the *AutomaticAccess* MSC (compare to *composite MSC*). Choosing the corresponding menu item (`LeaveMessage`), the user is asked to leave a message. In addition, the MSC illustrates that use cases can be linked via references, a standard construct in the MSC-96 standard. The (menu) choice construct is thereby implemented via a naming convention, where a use case reference with prefix *exc* identifies a conditional reference.

### 3.2 Formalisation in system analysis

Due to the earlier lack of tool support for object modeling, the first step was to translate the textual object model into OMT notation. Without tool support for object modeling a smooth transition between requirement analysis and design is considered to be impossible, as it prohibits the use of an intermediate formalism that states requirements in an abstract way. Having partly revised and redefined the object model, we got on the one hand a static view of the Voice Mail service (shown in Figure 4), in which object classes are related by *consists of* relations. On the other hand we got a dynamic view of the service (shown in Figure 5), in which relations express information flow between object classes.

In system analysis the system is no longer viewed as a black-box. Requirements are distributed either on a newly defined architecture or, as in our case,

[3]Telelogic SDT 3. 1.

on an already existing one, the IN architecture. Figure 6 shows an example of a use case in the system analysis phase. The MSC instances are in fact object instances, i.e. they represent objects. To indicate this, the naming of the instances include both an object name and the corresponding object class name. Note that we assume that a process instance in an IN node triggers (can trigger) a process instance in the same or another node. For example to check for a legitimate B-subscriber, a process in the `SC_LN` (Service Control Logical Node) that is represented by an (active) object of type `AccessControl` triggers (by use of the operation `CheckBNr`) another process instance in the `SD_LN` (Service Data Logical Node), that on the other hand is represented by an (passive) object of type `SubscriptionDBType`. It should be noticed that not all MSC messages map strictly to class operations. In some cases the operation is synchronous, i.e. demands a return message (here e.g. `Ok`).

### 3.3 The design phase

The system design activity aims at producing a design architecture and to further refine the use cases in order to make them suitable for verification. Here we restrict ourselves to the object oriented design, that takes the analysis object model developed in the previous phase as input. Figure 7 gives a small hint, how an object design for the Voice Mail service (TeleBud) could look like in SDL. According to the analysis object model, the top class of the aggregation hierarchy (Figure 4) has been mapped to an SDL system. What cannot be seen in this figure is that the leaf nodes of the aggregation hierarchy have been mapped to SDL processes (process types) and the intermediate nodes to SDL blocks (block types). For a study of SDL and its object oriented features see [11]. The transition between the models was supported by the *Paste As* concept of SOMT, thus preserving (vertical) traceability. The following steps, refining SDL blocks and processes, defining (SDL) procedures, choosing data types etc. require lots of efforts and design decisions, but nevertheless are considered to be straightforward.

## 4 Summary

In this paper we tried to pave the way for a smooth transtion from requirements capture to design by combining the methods ARENA and SOMT. Experiments with a Voice Mail service showed how this combination could actually work. Our intention was to make the methods and their combination practicable for real-life service development. Although the experiments made were not so extensive, we clearly outlined the benefits of the combination of the methods with respect to traceability. In our view the great challenge for the future will be to provide tools able to handle a whole service development project from the requirements to test cases, or at least the co-operation of several tools equipped with interfaces to each other aiming for a smooth transition between the different phases in service development.

## References


[1] Björn Regnell, Åke Davidson. *From Requirements to Design with Use Cases - Experiences from Industrial Pilot Projects.* 3rd Intl. Workshop on Requirements Engineering, Barcelona, June 1997.

[2] ARENA draft report. *Requirements Engineering in a Telecommunication Environment.* Uppsala University, August 1997.

[3] Telelogic SDT 3. 1. *Methodology Guidelines Part 1: The SOMT Method.* Telelogic AB, Malmö. November 1996.

[4] ITU-T. Recommendation I. 312 / Q. 1201. *Principles of Intelligent Network Architecture.* October 1992.

[5] Dirk Auchter. *Tool Support for Requirements Engineering: Applying the ARENA methodology.* Master of Science thesis, April 1997.

[6] Lennart Månsson. *Use of formal methods in Service Development at Telia.*

[7] J. Rumbaugh et al. *Object-Oriented Modeling and Design.* Prentice-Hall International, Inc., 1991. ISBN 0-13-629841-9.

[8] ITU-T. Recommendation Z. 100. Specification and Description Language (SDL). 1994.

[9] ITU-T. Recommendation Z. 120. Message sequence chart (MSC). In *Criteria for the use and applicability of formal description techniques*, September 1994.

[10] Telia Research AB. Np 93 090 Rev. B. *Kravspecifikation Röstbrevlåda*, 1994. Telia internal information, not externally available.

[11] A. Olsen et al. *Systems Engineering Using SDL-92.* Elsevier Science B. V., 1994. ISBN 0-444-89872-7.

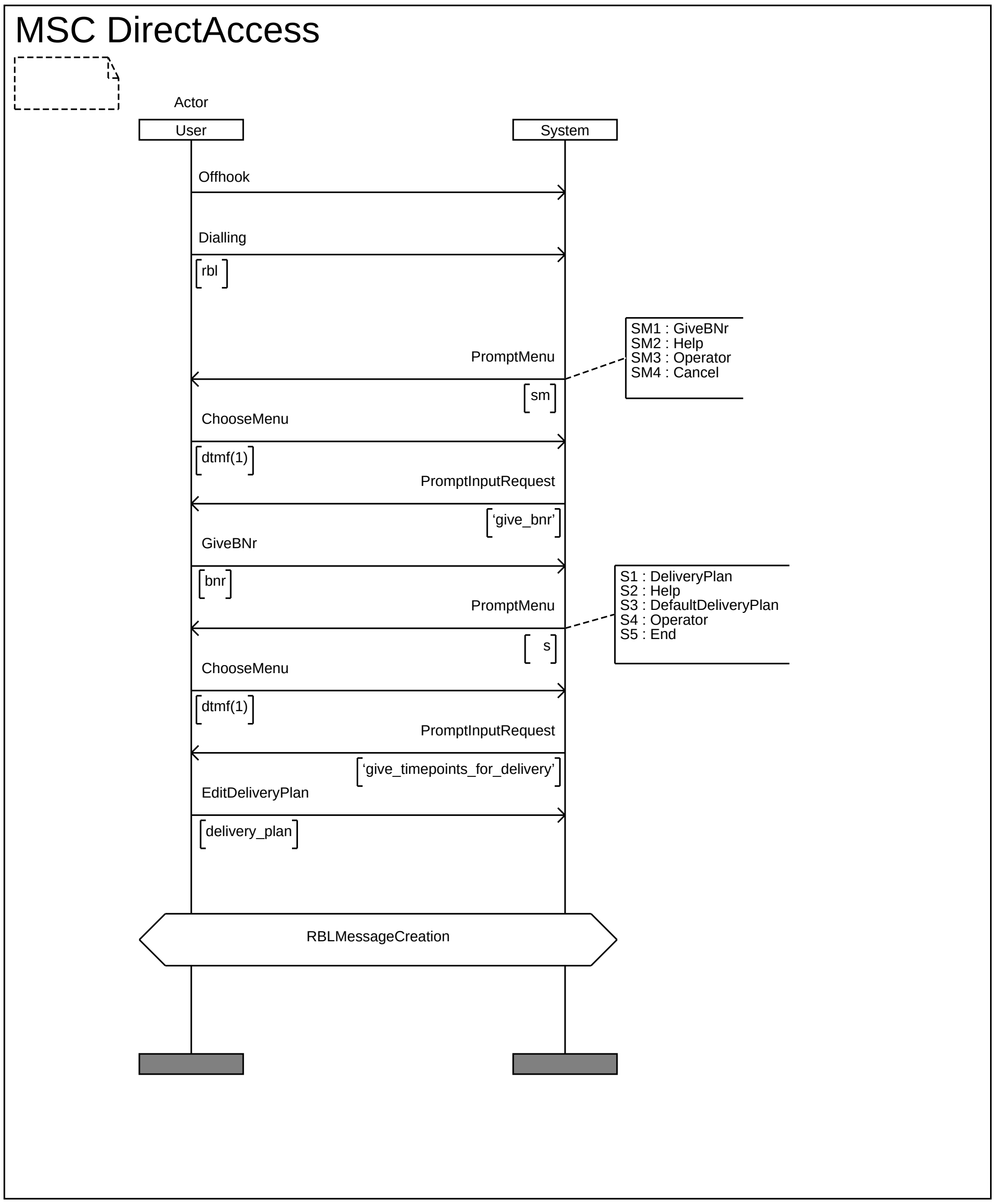


Figure 2: Voice Mail requirements use case (1)

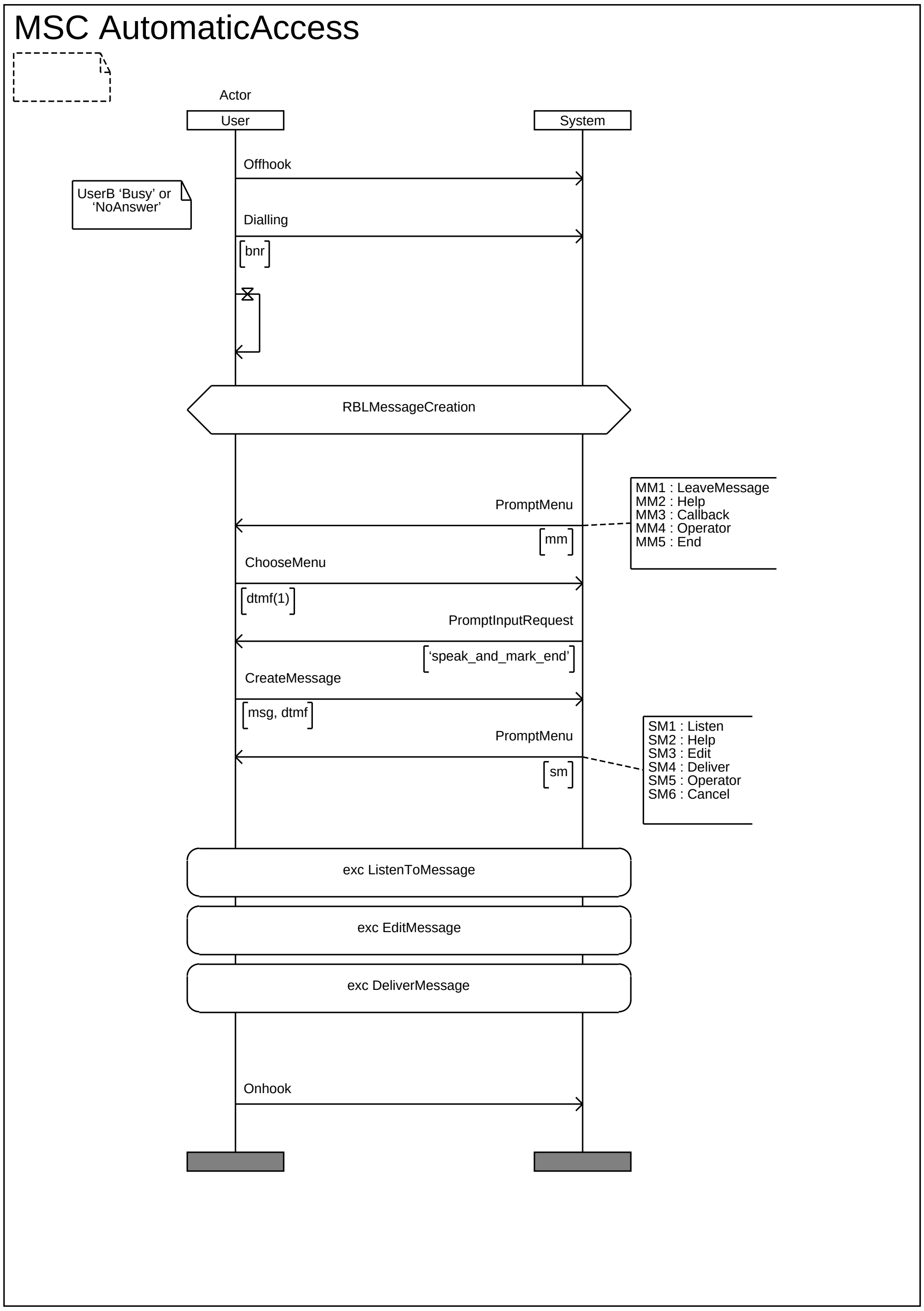


Figure 3: Voice Mail requirements use case (2)

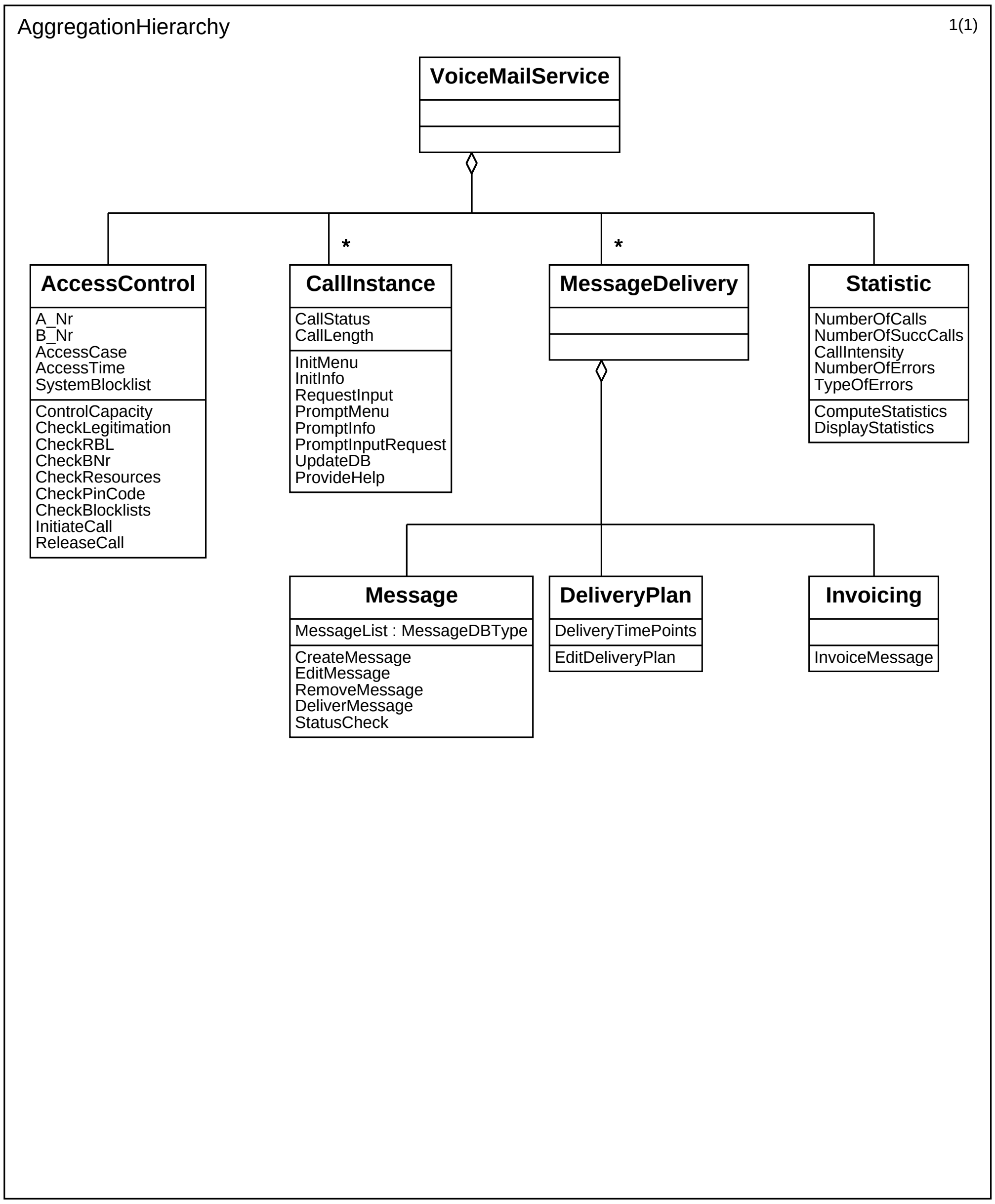


Figure 4: Voice Mail object model (static view)

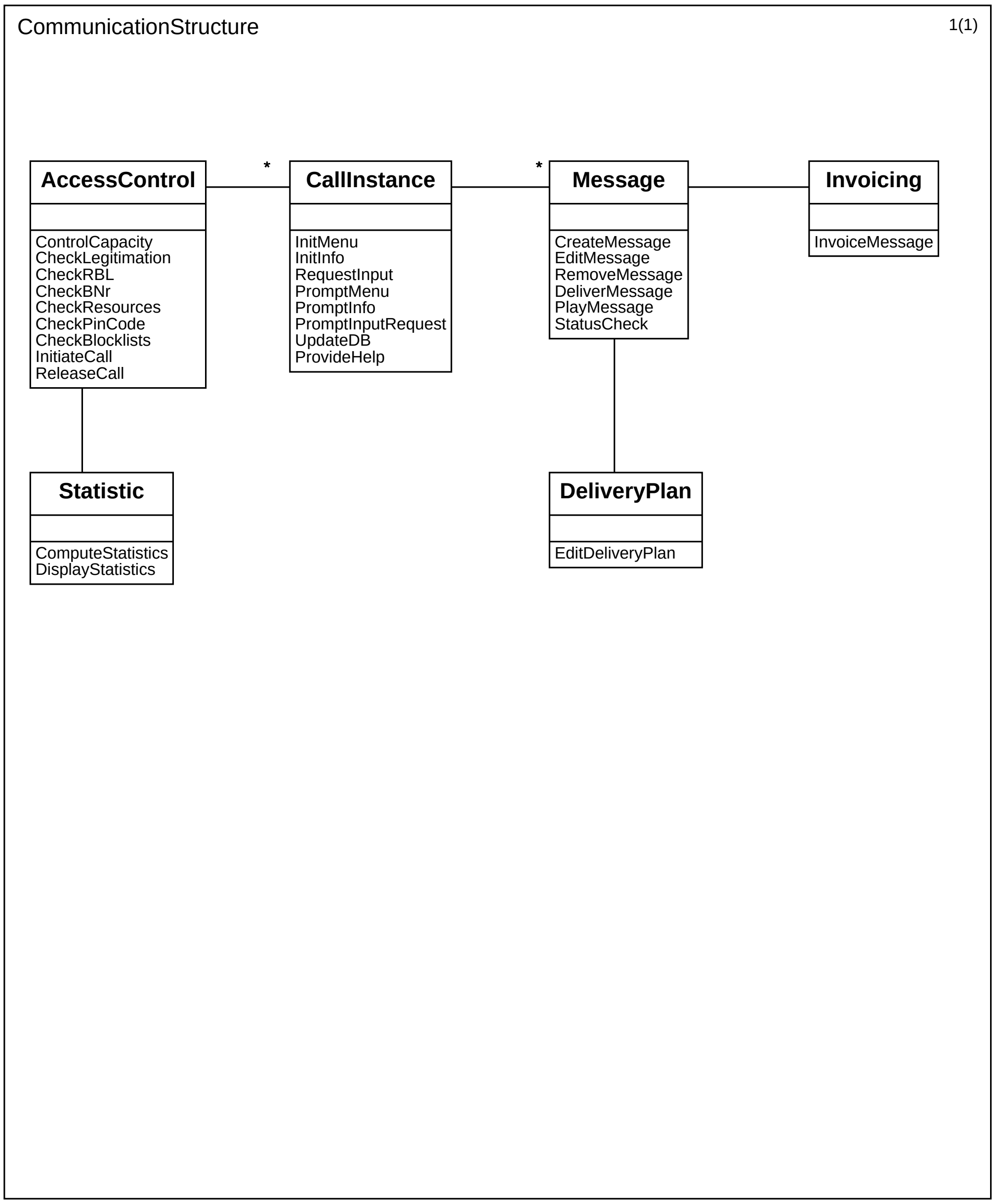


Figure 5: Voice Mail object model (dynamic view)

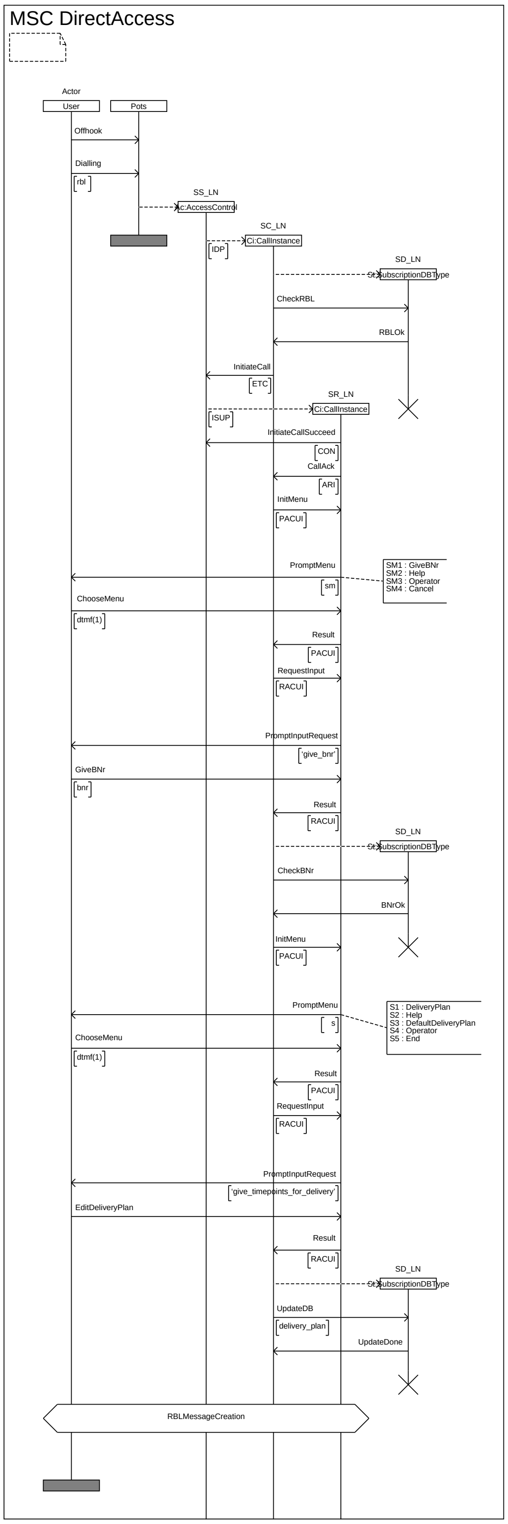


Figure 6: Voice Mail analysis use case

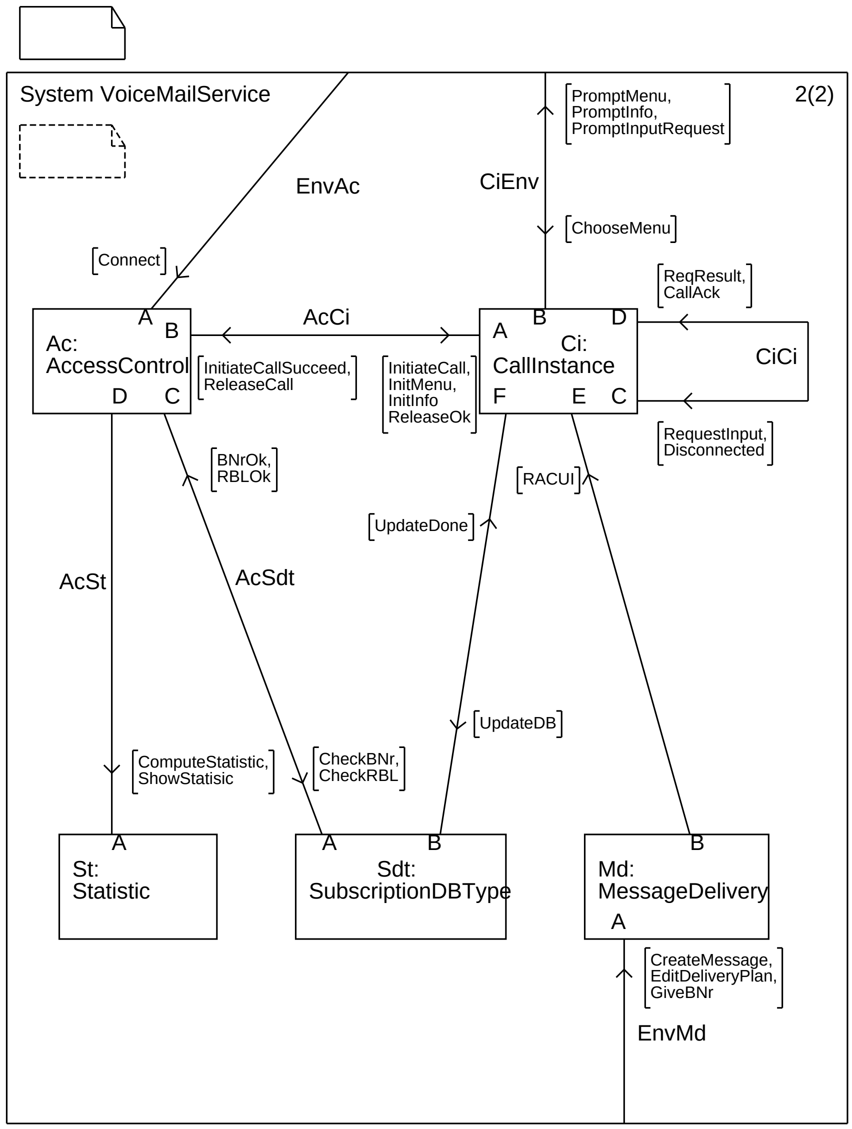


Figure 7: Voice Mail object design