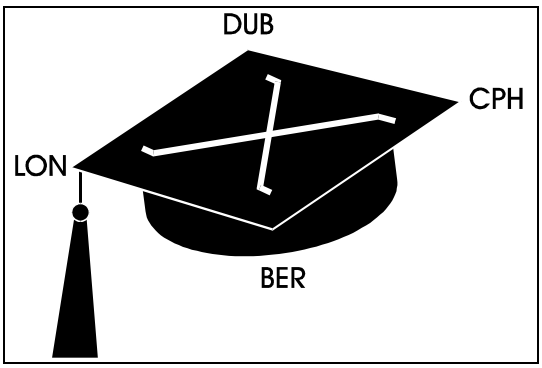
Project Title: **The PROSPECT of Multi Domain Management in the Expected Open Service Market**



Title of Document:

Prospect Methodology for the design of Multi Domain Management Services in an Open Services Environment (Trial 2)

Document Number:	AC052/TCD/WP2/040
Date of Current Draft	*Feb 25th 1998*
Contractual Date of Delivery to CEC	*February 28th 1998*
Actual Date of Delivery to CEC	
Editor	Vincent Wade, Trinity College Dublin
Deliverable Type	I–Internal Project Usage
Deliverable Nature	S–Specification
Workpackage Contributing	WP2

Abstract:

This document presents an overview of the design methodology for Prospect Trial 2. It is an evolution of the Prospect methodology used in Trial 1 for developing Multi Domain Management Systems in an Open Service Environment.

Keywords:
Open Distributed Processing, Inter-Domain Management Services, System Design Methodology, UML, CORBA IDL, Open Service Market, Multi Service Provider Environment.

Copyright

The Prospect Contractors are:

Organisation Name	**Short Name**	**Country**
Broadcom Éireann Research Ltd (PRIME)	BRI	Ireland
DELTA Danish Electronics Light and Acoustics	DLT	Denmark
DeTeBerkom GmbH	DTB	Germany
DSC Communications A/S	DSC	Denmark
GMD FOKUS	GMD	Germany
IBM Deutschland Informationsysteme GmbH	IBM	Germany
L.M. Ericsson A/S	LMD	Denmark
ATOS Ingenerie & Integration	ATOS	France
Tele-Danmark	TDK	Denmark
Trinity College Dublin	TCD	Ireland
University College London	UCL	United Kingdom
UH Communications Aps	UHC	Denmark
IONA Technologies	IONA	Ireland

List of Contributors

The following people have contributed to this document:

Vincent Wade [Editor] (TCD, Ireland), David Lewis (UCL, UK), Ralf. Bracht (IBM, Germany) Herve Karp (Atos, France).

Executive Summary

This document describes the rigorous design process developed in the Prospect project for designing and implementing multi domain service management systems. The design of multi domain management services is problematic as the development process needs to cater for: (i) multi domain management services (i.e. management services which span across a chain of several service providers), (ii) inter domain management services (i.e. management services which cross a single administrative boundary between two service providers), and (iii) intra domain management services (i.e. management services used within a service providers domain, but which may be reusable/customisable in more than one domain e.g. subscription, accounting).

Several methodological approaches are possible for such management services, and this document identifies the applicability of these approaches and the current trends in development of distributed management systems. The document then presents the context in which multi domain management services are developed and describes the design process developed in Prospect for such management services. The approach taken in the design process is not to re-invent modelling techniques, but rather to choose the most appropriate modelling techniques and harmonise them within the Prospect design process. The modelling techniques used in the design process are drawn from the Unified Modelling Language (UML) and TINA-C modelling activities.

The document illustrates the use of the design process by describing the design process for a multi domain management service based on components the Prospect trial; namely the Prospect Customer Management System. The example usage of the design process depicts the multi domain, inter domain and intra domain (re-usable) components of the management services.

Finally the document concludes with observations and experiences in applying the design process within the Prospect project and in using modelling tools to support the design process.

This page intentionally left blank

1. INTRODUCTION....1

1.1 OBJECTIVES OF THE PROSPECT MULTI DOMAIN DESIGN PROCESS.... 1

1.2 OVERVIEW OF DOCUMENT.... 1

2. MOTIVATION AND REQUIREMENTS OF A MULTI DOMAIN SERVICE MANAGEMENT DESIGN PROCESS....2

2.1 COMPONENT REUSE.... 2

3. TRENDS IN SYSTEM DESIGN FOR TELECOMMUNICATION NETWORK & SERVICE MANAGEMENT....3

3.1 TENDS IN OBJECT ORIENTED SYSTEMS DESIGN.... 3

3.1.1 Modelling and Designing Multi Domain Systems....3

3.2 APPROACHES TO REUSE OF PRE-EXISTING COMPONENTS AND THE DEVELOPMENT OF RE-USABLE COMPONENTS.... 4

4. PROSPECT MULTI DOMAIN SYSTEMS DESIGN PROCESS....5

4.1 MODELLING NOTATIONS AND METHODOLOGICAL TECHNIQUES.... 7

4.2 OVERVIEW OF PROSPECT DESIGN PROCESS.... 7

4.3 MODELLING/DESIGN TECHNIQUES ADOPTED FOR PROSPECT TRIAL SYSTEMS.... 9

4.4 SPECIFICATION TECHNIQUES ADOPTED FOR DESCRIPTION AND DOCUMENTATION OF PROSPECT TRIAL SYSTEMS.... 10

4.5 GENERATING ODP VIEWPOINT DESCRIPTIONS.... 11

5. EXAMPLE OF MULTI-DOMAIN SYSTEM ANALYSIS: CUSTOMER MANAGEMENT TRIAL....12

5.1 ENTERPRISE MODEL.... 12

5.2 USE CASES.... 14

5.2.1 Use Case: Accesses TES Service....15

5.2.2 Use Case: TES Customer Administrator sets up specific SAG intended for TES end users....16

5.2.3 Use Case Breakdown....16

5.3 SYSTEM ARCHITECTURE.... 18

6. EXAMPLE DESIGN OF INTER DOMAIN SYSTEM USING PROSPECT DESIGN PROCESS: TELE-EDUCATION SERVICE MANAGEMENT....19

6.1 ENTERPRISE MODEL.... 20

6.2 USE CASES.... 20

6.2.1 Subscribe TES Customer....21

6.2.2 Get Customer Bill....21

6.3 OBJECT MODEL.... 22

6.3.1 Subscription Registrar Propagator....23

6.3.2 Customer Operator User Session Manager....24

6.4 SEQUENCE DIAGRAMS.... 25

6.4.1 Subscribe TES Customer....25

6.4.2 Get Customer Bill....26

7. EXAMPLE OF DESIGN OF RE-USABLE COMPONENT USING PROSPECT DESIGN PROCESS: SUBSCRIPTION MANAGEMENT 27

7.1 ENTERPRISE MODEL 27

7.2 USE CASES 27

7.2.1 Subscribe Customer.....28

7.2.2 Authorise User Group.....28

7.3 OBJECT MODEL..... 29

7.3.1 Subscription Contract.....30

7.3.2 Subscription Registrar 31

7.4 SEQUENCE DIAGRAMS 32

7.4.1 Subscribe Customer.....32

7.4.2 Authorise User Group.....33

8. CONCLUSIONS 34

8.1 CHOICE OF NOTATION LANGUAGE AND SPECIFICATION TECHNIQUES 34

8.2 TOOL SUPPORT FOR UML 34

8.3 CONCLUSION ON DESIGN PROCESS..... 35

8.4 POSSIBLE FUTURE ENHANCEMENTS FOR THE PROSPECT DESIGN PROCESS 35

GLOSSARY OF DEFINITIONS USED IN PROSPECT DESIGN PROCESS.....36

REFERENCES 40

The Prospect Methodology for Design of Multi-domain Management Services in an Open Services Environment

1. Introduction

With the deregulation of telecommunication network services in Europe, there is increasing interest in telecommunication services being offered by third party service providers with network services being provided by different telecoms operators. The benefit of such an open service environment would be in providing the 'one-stop-shopping' delivery of 'tailored' services to end customers without these customers having to deal with the multiplicity of underlying telecommunication services and network providers.

The difficulty with such an environment is the complexity of managing the services across the different provider organisations (administrative domains). The design and development of integrated, co-operative management services spanning individual telecommunications service and network providers, value added service providers and final end users can be both complex and very time consuming.

In order to assist the design and implementation of such management services (called 'Multi Domain Management Services' because they cooperate across multiple network and service providers), Prospect has developed a design process. This design process facilitates the design and development of individual management services (within a providers domain) and their co-operation and interoperation across the value chain of network and service providers. The design of such management systems should not ignore existing standard component designs. Therefore the Prospect design process facilitates the re-use of management component designs and implementations based on the TINA-C results. However the methodology itself is independent of these results.

1.1 Objectives of the Prospect Multi Domain Design Process

The objective of the Prospect System Design Process is *to enable the design, development and implementation of an integrated, coherent Multi Domain Service Management systems.* This design process faciliates the analysis of the Prospect problem domain and supports the design and implementation of Multi Domain Service Management systems. The Prospect design process addresses the issues in developing end-to-end multi domain management systems. Prospect applies standard modelling and specification techniques where applicable and developes a design process in which these techniques could be applied for multi domain management system development.

1.2 Overview of Document

This document first outlines the motivation for the development of a design process for describing, specifying and developing inter-domain management services. Such management services have specific requirements since they span several administrative, technological and organisational boundaries. The document then outlines the current developments in design techniques and notations and clearly establishes the starting point for the Prospect methodology. Chapter 4 describes a high level view of the design process and identifies the modelling notations adopted by the Prospect design process. In order to illustrate the application of the design process. Chapter 5 describes the design of a Multi Domain Management Services, namely customer management. These management services cross several organisational boundaries in an 'end-to-end fashion'. For example, management operations performed by the Customer Management system (on the customer site) cause interactions with the value added service provider management system (in Prospect case this is a Tele-Education Service Provider management's system). These interactions, in turn, cause interactions with the Multi Media Conferencing Service

Provider and Hyper Media Service Porvider management systems. Hence Customer Management operations cause a 'chain' of interactions across the various provider management systems. In order to illustrate the application of the design process, chapters 6 and 7 describe example Prospect system designs developed using the design process and notations. Chapter 6 describes the Tele-Educational Service Management. . This management service has to co-operate and integrate with several management systems across administrative boundaries e.g. Virtual Private Network (Management) service. Chapter 7 describes the Subscription Management Service which is re-used in several Prospect Provider domains. Chapter 8 presents conclusions concerning the design process and its application in theProspect system. This chapter also includes the developers experience with software tools which supported the notations used in the Prospect design process.

2. Motivation and Requirements of a Multi Domain Service Management Design Process

The objective of the design process is the development of Multi Domain Management Services. Multi Domain Management Services are management activities, which by necessity operate across different management administrations or organisational boundaries. Multi Domain Management Services are vital if multiple value added service providers are to be able to manage and add value to various underlying tele-services offered by different network or service providers, e.g. a tele-educational service provider building an educational service using a Virtual Private Network from a network provider. Therefore the design process used to develop such management systems, must capture the interrelation and co-operation of the management services across different domains as well as supporting the development of management systems within an organisation (administrative domain).

Rather than developing new modelling techniques and notations, the design process should select appropriate standard modelling techniques and integrate these to provide a design process for the multi domain management systems. As the Open Distributed Processing standards in telecommunications are a well established way of describing system designs from various perspectives, the design process should facilitate the generation of ODP based viewpoint models for system description rather than attempting to define design steps which interlink each viewpoint model specification explicitly. It should also facilitate the introduction and integration of pre-existing models, designs and components into the system under construction e.g. TINA based component specifications.

The design process should use standard object oriented design notations for design visualisation and specification and should support an iterative, architectural based design process. More specifically it should comprise elements which (i) identify the stakeholders (e.g. organisations), responsibilities, roles and activities which form the business aspect of the management service being developed (ii) provide support for description of scenarios or 'use cases' (iii) provide a means of integrating pre-existing designs and specifications into the system model under development (iv) provide a notation for the identification and specification of object classes, interfaces, relationships and interactions (v) provide a mapping of objects into components for implementation and possible distribution (vi) support a way of planning, performing and executing implementation and infrastructure technology testing

2.1 Component Reuse

In the Open Service Market component reuse will occur more frequently between different organisations. There is a growing move towards the use of off-the-shelf components in the development of telecommunication software systems as (public) network operators are outsourcing management system development and are developing less bespoke systems [Adams-96]. The trend is toward the adoption of individual (packaged) systems and components, and the integration of these systems to form solution sets. This is motivated by the need to reduce costs by buying in (rather than building components from scratch) and improve quality by using reliable, robust components.

The design process should therefore cover both the development of the systems, using reusable components, and the development of the reusable components themselves. These in fact should be very

similar in that they will both take advantage of the same development principles, and that their concepts, notations and conventions should be similar in order that components can be easily integrated into the design of systems.

3. Trends in System Design for Telecommunication Network & Service Management

3.1 Tends in Object Oriented systems design

The late eighties and early nineties has seen the rise in usage of many different object oriented analysis and design techniques. Principle among the 'second generation' of these methodologies are Rumbaugh's Object Modelling Technique (OMT) [Rumb-91], Ivar Jacobson's Use-Case driven OO software Engineering Model [Jacob-92] and Grady Booch's Object Oriented Analysis and Design (OOAD) methodology. Another more recent object oriented methodology has been the FUSION methodology developed at HP.

The current trend in object oriented notation and distributed system specification is to harmonise existing approaches rather than develop brand new modelling techniques. An example of this trend is the proposal of a Unified Modelling Language (version 1.0 released in March 1997). UML is a set of modelling notations, which are independent of any software development process. It specifies the modelling notation and the semantics underlying this notation. UML has utilised and extended modelling elements from Rumbaugh's OMT, Jacobsen's Software Engineering Methodology and Booch's OOAD methodologies, as well as other lesser known modelling techniques. The notations are claimed to be backward compatible to these three principal modelling approaches [Fowl-97]. Also the Object Management Group (OMG) - the consortium responsible for CORBA standards - have recently called for proposals for standard modelling notations (closing late 1997). UML itself is independent of any particular design process. It is envisaged that design processes will be tailored for specific application domains. Therefore there is no overall 'best' design process because design processes for different types of systems tend have different foci and emphasise different characteristics e.g. the design process for an office system would be quite different to that of a hard real-time system [UML-97].

Since Multi Domain Management Services inherently require distributed solutions, an open distributed approach must be adopted for multi domain *system description*. Currently several standards address the issue of specifying open system. Principle among these is the Open Distributed Processing [ODP-94] standard that suggests the specification of five different viewpoints of the system (Enterprise, Information, Computation, Engineering and Technology). These viewpoints highlight, and provide a basis for the separate description and discussion of, different aspects of the design and implementation of the system. Several standards have taken these viewpoint description concepts and enhanced them for describing network and service management systems e.g. TINA [TINA-93], [TINA-95] and ODMA. However, the ODP viewpoints do not provide a prescriptive design process that can be followed for developing and implementing management systems [ITU-96].

The ITU-T Telecommunication Management Network recommendation [TMN-95] includes a recommendation titled 'TMN Interface Specification Methodology' [M3020-94] which provides guidelines for the functional decomposition of management interfaces which occur within the TMN functional architecture. This methodology does not provide any direct guidelines for the development of management systems internals, but rather provides a path for arriving at specifications for the interfaces providing management functions (expressed in GDMO [X722-92]). The approach in M3020 is intended principally for the specification of Standard MIBs for network elements using the CMIP network management protocol.

3.1.1 Modelling and Designing Multi Domain Systems

In order to capture the idea that (management) services can be used independently of each other as well as being used in cooperation, the end-to-end management system must be viewed in two complementary (overlapping) models: (i) Inter Domain Management Model: This models the management services in co-

operation to support end-to-end management services. (ii) Intra Domain Service Management Model: These models describe the management services (e.g. configuration, accounting etc.) of each of the tele-services in an Open Service Market e.g. management services for VPN, MultiMedia Conferencing, HyperMedia Services.

The reason for this decomposition is that each management service could exist on its own (i.e. is a usable management service regardless of its cooperation with other services management providers). This reflects the view of the Open Service Market where the management of a tele-service has its own objectives whether or not they are infact integrated across provider organisations Thus inter domain models capture the end-to-end management chain while the intra domain models capture the management services of each individual service in isolation. However, these models are not completely independent as components used for inter domain management are also used in the intra domain management.

3.2 Approaches to reuse of pre-existing components and the development of re-usable components

Experience in Prospect and other projects related to multi domain management [Lewi-95] suggests that relying on only an interface definition to a component, supported by plain text descriptions for the behaviour of the objects defining the interface, is inadequate for describing the component's behaviour. This seems to be especially true when operations at the interface trigger interactions with further components, rather than simply providing access to internal data representations. The problems caused by having to rely solely on the quality of the behaviour descriptions in interface definitions can be alleviated by accompanying documentation which indicates the dynamic behaviour of the interface with relation to other component interfaces, for instance through sequence diagrams. This reflected in the approach taken in NMF Ensembles [NMF-92].

In addition, however, the rapidly changing requirements imposed on systems by the open service market means that component reuse will often involve augmentation of the component itself. However to modify a reusable component correctly and effectively, i.e. by taking advantages of its initial robust object oriented design, the underlying design model behind the component's interface must be accessible. ODP attempts to address this by accompanying the computational viewpoint model by an information viewpoint model. However experience in Prospect in using TINA components seems to indicate that this descriptive technique is also inadequate, primarily due to the lack of a formal mapping between objects in these two viewpoint models. In the case of Prospect, this was overcome by reverse engineering use cases for the computational model, in order to illuminate this mapping, however this still only provided an implicit rather than explicit mapping between the two viewpoints.

The software engineering community has recognised similar problems with reuse that is based solely on class definitions. One response to this has been to take the concept of Design Patterns from the architectural and construction industry and apply it to software reuse as demonstrated by Eric Gamma [Gamm-95]. Design patterns aim to try and capture the knowledge of experienced designers and document it in a way that facilitates the communication of architectural knowledge and of known design traps to other developers. Typically design patterns represent common, well-proven designs. There are several different forms suggested for design pattern documentation, however they all follow a common structure. At a minimum a design pattern states a problem and outlines a solution to it, with a context description that indicates the applicability of the solution. The latter part is key, since the aim is not to sell the pattern to the reader, but to provide the information needed by the reader to enable them to gauge whether the solution presented fits well to a problem with which they are faced and satisfies any other (possibly non-functional) requirements placed upon a solution. An important feature of a pattern is giving it a suitable, and preferably brief, name. In this way it is hoped that pattern based terminology will evolve into a more powerful means for communicating between software developers. Great emphasis is placed on expressing patterns concisely and clearly and ideally they should also contain an example of an application or coding of the pattern.

Design patterns are typically collected together into Catalogues [Copl-97], though more benefit can be gained for the user when a collection of related patterns, possibly addressing a specific application domain, are carefully cross-indexed, to show how the solutions can work together in different ways. Such an inter-related collection of patterns is referred to as a Pattern Language.

Gamma's original pattern catalogue addressed how small groups of software objects addressed common problems, however patterns have been written to address a wide range of problems up to and including patterns for organisational structure. Mowbray and Malveau [Mowb-97] suggest that patterns could actually be categorised into the following levels of architectural scale:

- Global, which contains patterns that span several organisations.
- Enterprise, which contains patterns applied across a single organisation.
- System: which contains patterns concerned with the structure a system consisting of a set of applications.
- Application; which contains patterns addressing specific user applications.
- Macro-Architecture, which contains patterns related to application frameworks.
- Micro-Architecture, contains patterns that present object models that solve common software problems (similar to Gamma's patterns).
- Object, contains patterns for reusable objects and classes.

In the context of management system design, some parallels can be drawn between the NMF's Ensembles and design patterns. Both attempt to address specific problem, and both package together the problem statement, the context and the solution. Ensembles however are aimed at solutions expressed in terms of interactions at a management interface, while patterns have unrestricted scope. It can be seen therefore that by presenting Ensembles as patterns, they could become part of a wider pattern language addressing the problems of telecommunications management system development in general. Structuring such a pattern language using the levels of architectural scale described above would allow the integration of patterns beyond those in existing Ensembles, which would most likely be located at the application level. This would provide a framework for collecting together patterns that related not just to reuse of specific interface specifications, but also ones which addressed both problems in constructing the systems behind the interfaces and how these systems would fit together in a multi-provider context. Though this use of patterns could be useful within a provider organisation's development effort, its application in the standardisation and component development areas of the telecommunications management software sector would be novel.

For the developers of reusable components, expressing their components, and their relations to other components and frameworks clearly, would benefit their customer by easing the process of selecting components from a potentially wide number available. The relation of component to standards could be clarified by treating them as patterns with points of conformance, rather than simply profiles of existing standards.

For the developers of open standards, a similar benefit would be passed to their "customers", in that management system and component developers could more readily compare and select standardised solutions. However it is not expected that patterns could replace standards, since the latter are still needed to provide consistent, and unambiguous definitions of interfaces. Instead, however, patterns may make the aim and application of standards more accessible than their current, necessarily complex and detailed form.

4. Prospect Multi Domain Systems Design Process

For the Prospect design process to be applicable to the development of systems in the Open Services market, the requirements of the different sector players, and their different resulting foci in using the process need to be understood. The potential users of the development process can be divided into three types of roles:

1. Sector players interested in analysing multi-domain management scenarios and solutions. This group includes various standardisation bodies such as the ITU-T, ETSI, IETF, TINA-C and the NMF, as well as industrial groupings attempting to come to an agreement on inter-working mechanisms for their information systems.
2. Players responsible for developing systems for service providers, these either being the service providers themselves or independent software developers subcontracted to perform these tasks.
3. Players that develop and sell general management components and management platforms for use by others.

Figure 4.1 represents the market sectors occupied by the different role types and their relationship to each other. The sector concerned with standardisation aims to generate industry agreement on architectural models and interfaces for use in the development of systems and components. For component developers this reduces the risk involved in developing a component. For system developers this enables easier integration with other systems. The system developer benefits from using the components generated by the component vendors while providing requirements both to component developers and the standards developers. The component developer also provides requirements to the standardisers.

The development process in Prospect was based on experiences where this separation of sector concerns was reflected in the working structure of the project. The project worked as a group to identify the stakeholders in its multi domain management scenarios, the management functions that would be required and the architecture that would be used for inter-domain interworking. Individual partners or groups were then assigned responsibilities for developing individual provider systems, management subsystems and reusable components. The overall development methodology ensured that these different development strands were consistent and the systems and components developed could then be readily integrated.

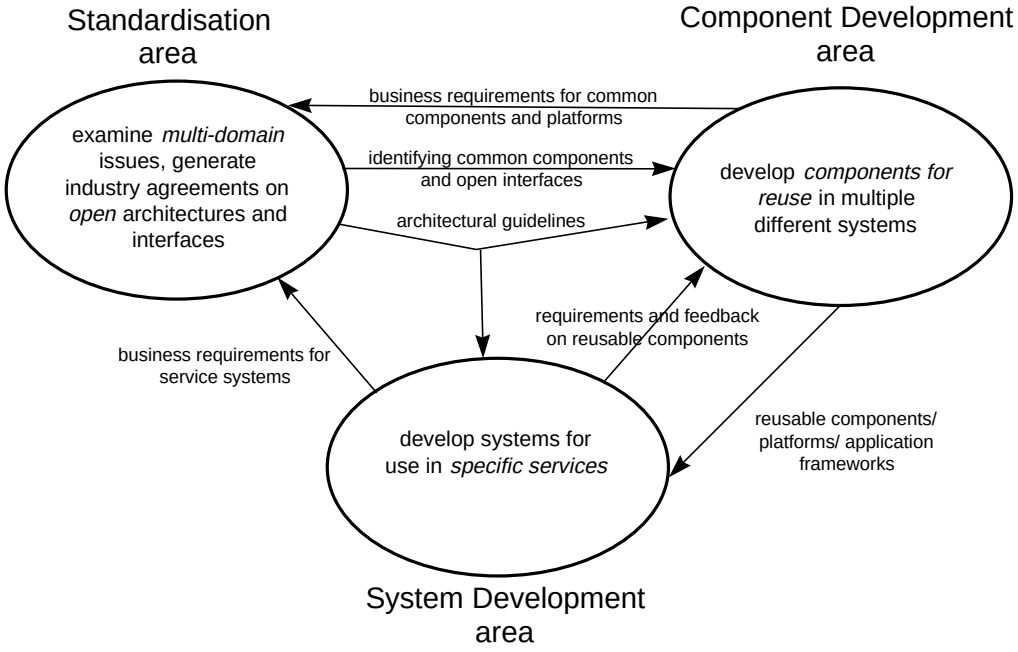


Figure 4.1: Areas of concern in multi domain management system development

4.1 Modelling notations and Methodological Techniques

Rather than developing new modelling notations or systems specification techniques, the Prospect design process harnesses existing modelling notations and concepts (UML) and specification languages (CORBA IDL). These notations and specification languages facilitates the description and specification of the inter and intra domain management systems The methodology also has been influenced by the modeling work of previous ACTS projects PREPARE [Hall-96] and PRISM [Berq-96] and the TINA-C Service Architecture [TINAC-94] and modelling approaches.

The design process supports each stage of development. Thus the Prospect design process iterates the required stages of system development which includes requirements description, business model description, object modelling, interface specification, implementation, deployment and testing. The ODP viewpoints provide a well-established approach to system model description. However, they do not provide a prescriptive methodology that can be followed in developing management systems [ITU-96]. Therefore unlike previous design methodologies which have attempted to specify design steps which allow the generation of one ODP viewpoint from a previous viewpoint, the Prospect design process sees the development of a full object oriented model as central to the overall design effort.

4.2 Overview of Prospect Design Process

The Prospect design process is concerned with the analysis, development, implementation, testing & trailing of inter and intra domain management systems. Figure 4.2 presents the design process as a cycle and identifies the different influences on the design process, the stages within the process and the specification that can be generated during the design process. The process is iterative and all stages need not be addressed on the first iteration of the cycle. However, in the first iteration requirements analysis (identification of stakeholders, responsibilities, use cases etc.), selecting the target architecture and planning for integration, testing and evaluation should at least be performed.

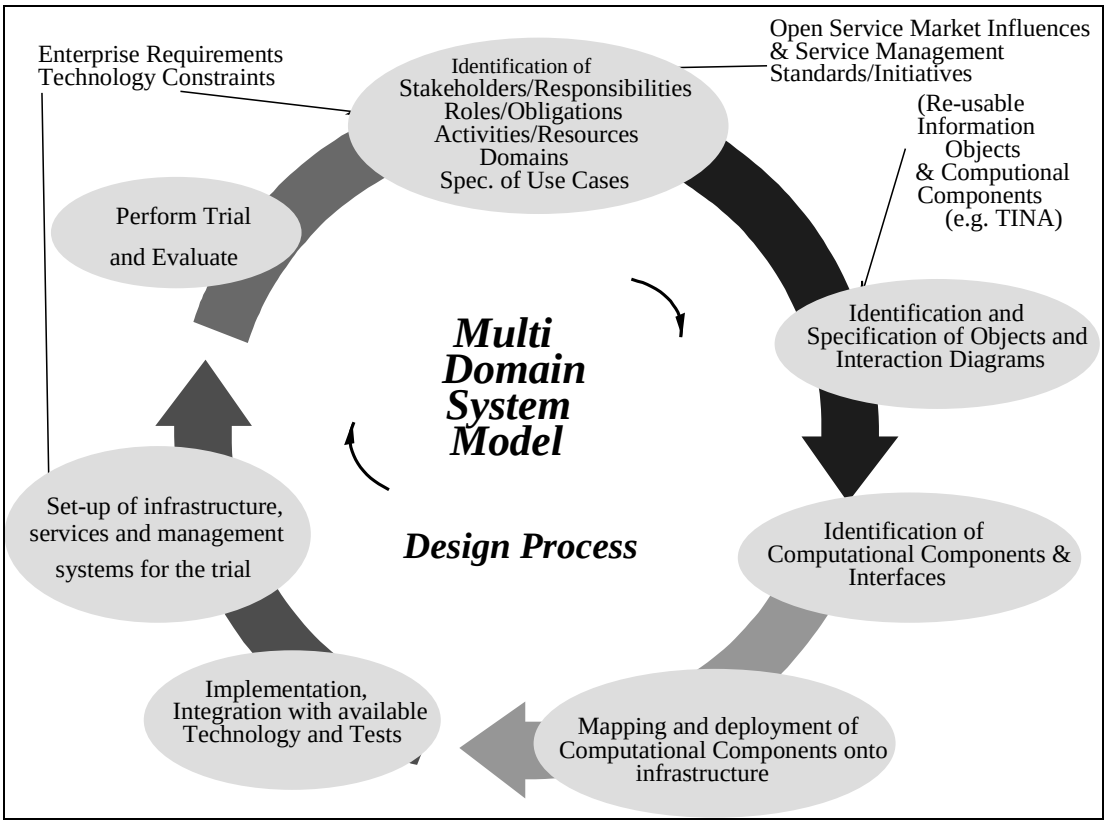


Figure 4.2: Prospect Design Process

The design process provides a structured way of developing, implementing and testing these object designs. It specifies the design steps for

- developing multi domain business models (which includes the representation of stakeholders, assignment of responsibilities, identification of obligations and activities etc.),
- use case definition analysis,
- object identification and relation representation,
- definition of new computational components and the integration and extension of pre-existing computational components (e.g. from TINA C Service Architecture),
- distributed deployment of computational components,
- definition of platform architecture (nodes) and platform services,
- generation of test sets,
- trial execution.

The Prospect design process also identifies key places in the development process from which specific ODP viewpoint models can be generated and prescribes the contents of each viewpoint model specification and describes how these can be generated. The information needed to generate these viewpoint specifications is a subset of the information needed to design and implement the actual systems. The design process ensures the consistency between each stage of the system model development and therefore provides a means of tracing the interrelationships between the ODP viewpoints. The benefit of generating the ODP viewpoint specifications is that a clear separation of the different aspects of the system can be captured. However, for the Prospect System Model Specification, such ODP viewpoint descriptions were not generated, as the Prospect designs developed during the design process were considered sufficient to describe and document the Prospect Systems implementation.

4.3 Modelling/Design techniques adopted for Prospect Trial Systems

This section identifies the design techniques used in the specification of the various Prospect systems. Thus the design and specification of a Prospect system is described by the model notations listed below. It is important to note that there is only one model of each system, however these different descriptions allow the specific aspects and concerns to be readily presented. UML modelling notations have been extensively used by the design process in the specification of the models.

The following design techniques and models have been used:

- Enterprise Models.
- Use Cases.
- Object Model(s).
- Sequence Diagrams.
- Definition of Components.
- Integration of Components into larger systems.
- Testing and Integration Techniques.

Design approach for Prospect re-usable components

In designing the 'components' to be implemented within each domain, it is important to ensure, as far as possible, the re-use of these components. Prospect has adopted a practical approach to the development of reusable components that has been focused on the integration of these components into the multi-domain systems being developed. This process has involved:

- Use cases specific to the component, including ones that show interactions with other components, as well as other systems. These can be summarised with use case diagrams.
- Component diagrams, principally showing the interfaces exported and the interfaces used by the components. These interfaces should map onto, either IDL definitions or some programming language API definitions.
- Sequence diagrams, again principally showing the dynamic interaction of external entities with the external interfaces of the component.

Approach adopted for integration and testing

These steps take place in the design process between the implementation phase and the deployment model, which defines the infrastructural set up for the trial.

The different steps to be conducted during this phase were:

1. preparation of integration and tests: this step starts at the beginning of the specification as its aim is to produce or reuse specification document(s) to support integration and tests. The documents to be created during this step are defined later in this section.
2. unit tests, which tested each component in "stand alone" mode; i.e. using stub interfaces and "hard coded" components (i.e black boxes).
3. a set of integration tests that is decomposed into intermediate integration tests between components and ending by a final integration of the system.
4. a set of acceptance tests, these tests are in relation with the Trial and are based on the use cases.

4.4 Specification techniques adopted for description and documentation of Prospect Trial Systems

The Prospect project is a 'trial based project' which means that one of the primary goals of the project was the trialling of multi domain management systems. Therefore Prospect applied the design process to support the following objectives:

- To support the development of systems, built largely from reusable components, that satisfy the specific goals of individual project trials.
- To support the evolution of common reusable components across the lifetime of the project (and possibly beyond).

The development of the trial systems involved the development of the models and descriptions listed below, during different stages in the development process. Figure 4.3 illustrates the stages, during the design process, where these descriptions were developed.

1. A trial enterprise model, identifying the stakeholders and roles involved in the trial. These are used as the basis for developing functional requirements.
2. A set of trial system level use cases, defining the overall aims of the trial system in terms of user interactions with the trial systems
3. A breakdown of the trial system level use cases onto requirements on the components and systems that make up the trial system.
4. A system architecture showing how the trial system is composed of individual components, systems and subsystems
5. A set of acceptance tests defining the interactions that need to be conducted to show that the full use cases can be conducted by the final system.
6. A set of integration tests that define intermediate integration tests between components. This should include references to inter-partner integration tests that are required between separately implemented parts of the component.
7. A deployment model showing how the trial system will be deployed for the trial itself.

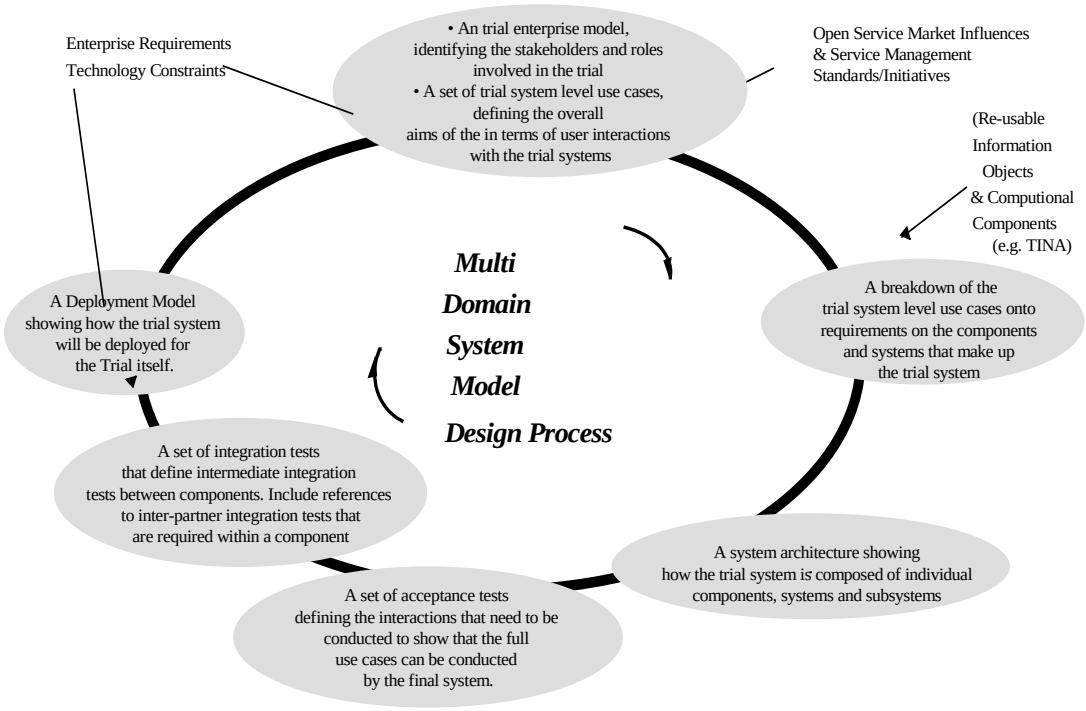


Figure 4.3 Elements of Trial System Descriptions

Description of re-usable components

In addition to these parts, which represent points required along the development path of the trial, additional documentation is required for each of the reusable components that are to be used in the trial. In these trials the evaluation of alternative reusable components has not been a major activity, since the components used where simply those available and few others were considered.

As most of these components are pre-designed to a large extent (from sources such as TINA), it was not feasible, or indeed necessary, to re-document all their development stages in the same manner as for the trial systems. However the component documentation needed to provide the following in order to facilitate the integration of the components into the system design:

- A set of generic use cases for the component. It should then be possible to map the requirements of the trial systems and individual stakeholder systems to these use cases to ensure that the component satisfies the requirements of these systems.
- A description of the component in terms of the interfaces it exports, which may be protocol interfaces, IDL interfaces or some other form of API-like interface. This component level description should also indicate which other components this component depends upon. These component descriptions may then be used in the system architecture models for the overall trial systems.
- A set of acceptance tests, traceable to the component use cases, that could be reused if necessary as parts of trial system acceptance tests.

The documentation was based as much as possible on the use of UML, using both Paradigm Plus and Rational Rose CASE tools.

Documentation of integration and tests

These steps are supported by a set of documents:

- Scenarios or test suites, defined from the use cases, that are used for the acceptance tests.
- Sequence diagrams, which is the base document for integration tests as it defines the interactions between components with defined parameters and values to be returned.
- Error cases which describe the exceptions to be returned in some situations.
- Naming conventions which allows the definition of constraints to be applied on specific parameters values. This has been found useful during integration tests.

Tools can be used during integration and testing stage of the design process, e.g. a common database for reporting defects.

4.5 Generating ODP Viewpoint Descriptions

It is not anticipated that ODP viewpoint specifications need always be generated but some management system developers may desire to use such ODP specifications to describe and compare their systems to external developers/managers. In order to facilitate the generation of such viewpoints, key elements of the models produced by the design process can be combined to serve as a basis from which Enterprise, Information and Computational ODP viewpoint models can be generated. The information needed to generate the ODP viewpoint specifications are a subset of the information needed to design and implement the actual systems. Thus the contents of each viewpoint model specification can be prescribed and generated from the elements of the system model.

Stage in Design Process from which Viewpoint is derived	*ODP Viewpoint*
Textual description of: - Stakeholders, Relationships, Roles, Obligations, Activities UML Class Diagram Notation diagrams used to represent the stakeholders and the relationships between them UML Use Case Notation (text based descriptions of user interactions) describing required operations/activities	Enterprise Model
UML Class & Object Model Diagrams, UML Interaction Diagrams (both sequence diagrams and Collaboration Diagrams) e.g. class, aggregation, etc. Specification of Class interfaces	Information Model
UML Component Model using IDL specification of component interfaces	Computational Model
UML Implementation Diagrams (showing components and deployment)	Engineering Model

Table 1 Mapping from elements of inter and intra domain models to ODP viewpoint models

In fact the ODP viewpoints descriptions (Enterprise, Information, Computation) were used in the early stages of the project but were not infact all needed in the final system description provided by the project [D4B-98].

5. Example of Multi-Domain System Analysis: Customer Management Trial

The target multi domain systems developed in Prospect are those used for performing user trials within the project. These trial systems were required to demonstrate the interactions of service and management systems between a number of organisations. The example presented here concentrates on the Customer Management Trial System developed in Prospect. The Prospect trial aims to demonstrate the design and implementation of management systems where 'chains' of interactions across the various provider management systems are required to deliver 'end-to-end' service management.

5.1 Enterprise Model

For each trial an enterprise model is required so that the inter-domain interactions can later be clearly identified, and so that the various functions provided by each organisation's system in the trial can be assessed in line with the organisations business objectives, e.g. providing a Tele-Education Service (TES) to a customer, or reselling multimedia services to the TES provider.

Enterprise models were expressed as object diagrams, where the objects were instantiations of the following stakeholder and role classes:

- composite provider class: a stakeholder providing a service composed of services provided by other service providers.
- multimedia teleservice provider stakeholder.
- value added service provider stakeholder.
- network operator stakeholder.
- provider administrator role.
- customer administrator role.

- end user role.

The relationship between these classes are shown in Figure 5.1 below. This set of classes was intended to cover the range of roles found in the various Prospect trials, however they are fairly general and may be extended to cover other enterprise situations related to telecommunications-based services.

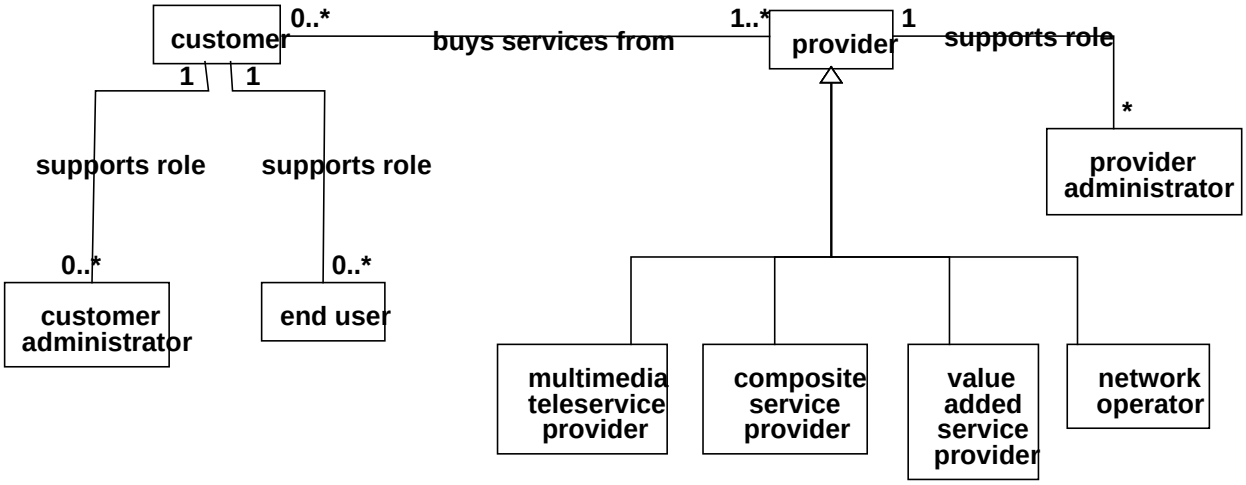


Figure 5.1 Prospect Trial 2 Enterprise (Class) Model

The enterprise model for a trial details the different organisations and user roles that were relevant to the overall trial system and the relationships between them. This was represented as a UML object diagram containing instances of the enterprise model classes defined for the project as a whole. The object diagram for the Customer Management Trial is show in Figure 5.2.

To provide a more detailed context for the definition of use cases, the exact state of the relationship between the roles and organisations prior to the trial is defined in the enterprise model. This take the form of statements of contractual relationships between the different stakeholders. Note that other relationships may be developed as part of the trial itself and the instantiation of such a relationship would therefore be the subject of one of the trial system use cases.

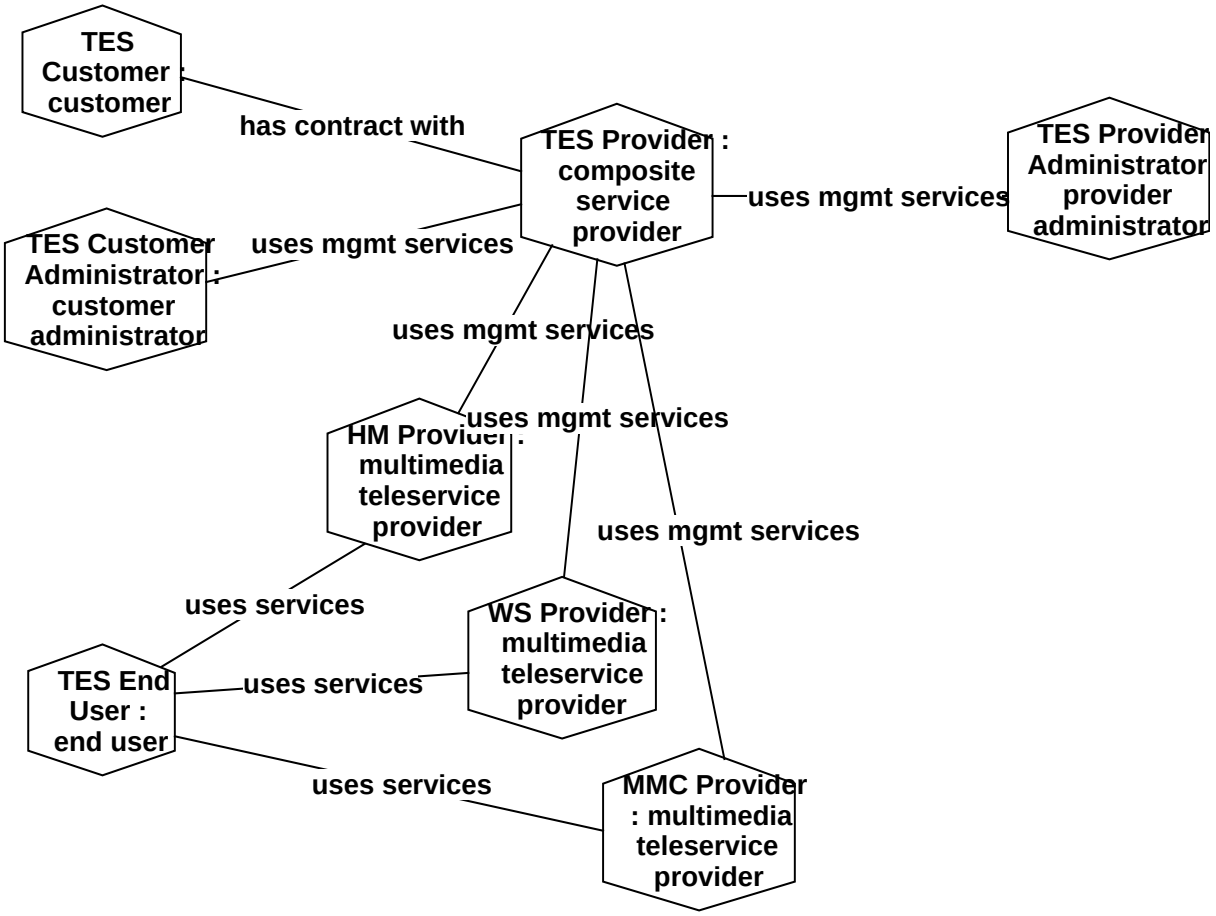


Figure 5.2: Customer Management Trial Enterprise Model

5.2 Use Cases

Trial system use cases express the user interactions which the trial should be demonstrating, thus providing the basis not only for the analysis and design of these systems that implement these use cases, but also the basis for the evaluation of the trial. These top level use cases are therefore stated in terms of the trial system as a whole and its interaction with actors that represent the users or evaluators of the trial itself.

The high-level use cases may be accompanied by an overview of how the use case will generate inter-domain information flows in order to help people understand the extent of the design impact of a particular use case, and should not be considered part of the use case itself.

UML use case notation can be used to summarise the relation between the use cases, and between the use cases and the actors involved. Individual use cases should however be defined separately giving the textual description of the use case, the preconditions assumed for the use case and possibly any specific open issues concerning existing designs raised by this use case.

The set of use cases and the details of those use cases changed over the development process, with factors from the later design and implementation stages always being fed back to the use cases so that they properly represented the functionality of the system. The use cases were summarised using UML as shown in Figure 5.3.

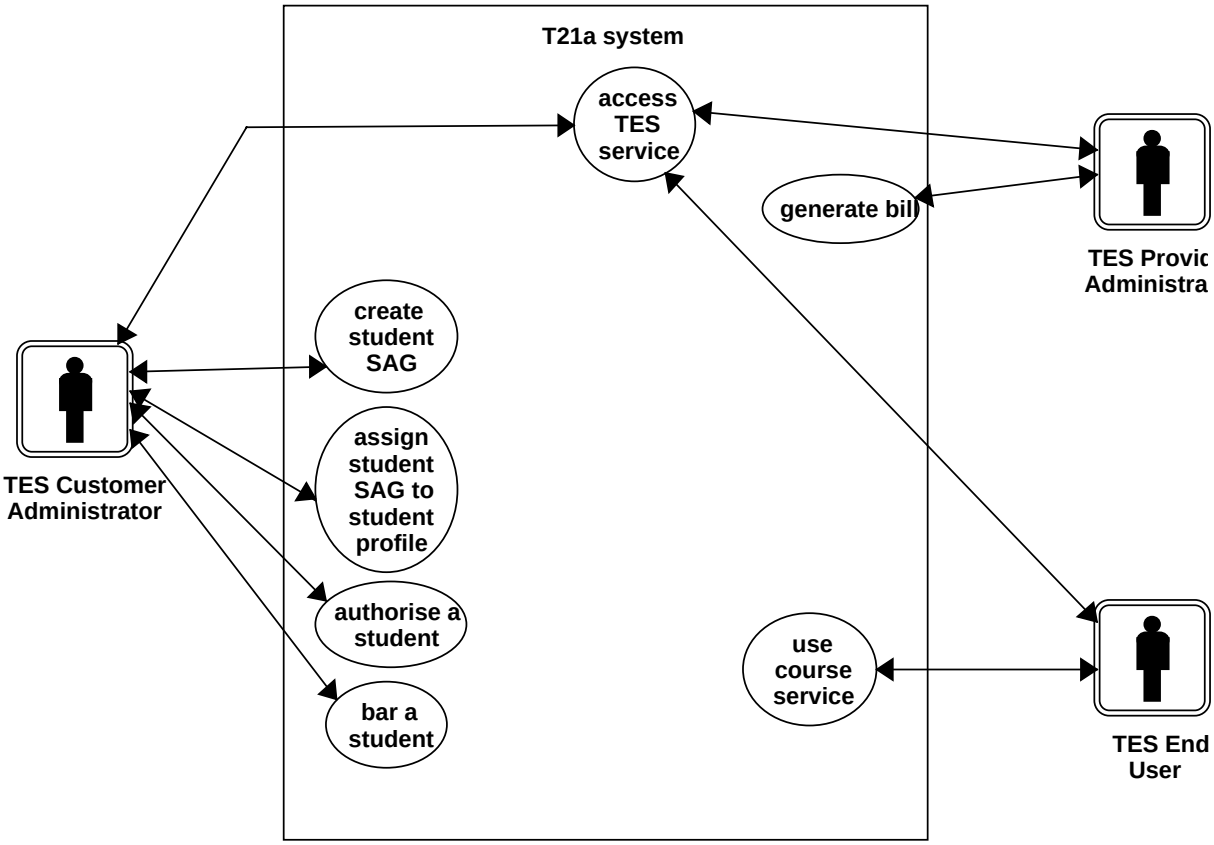


Figure 5.3: Use Case diagram summarising the Customer Management Trial system functionality

The use cases themselves were defined in plain text with a set of pre-conditions that aided in defining the sequence of use cases in relation to each other. Two examples from the set above are given below.

5.2.1 Use Case: Accesses TES Service

Preconditions:

The customer organisation has an account with the TES provider.
The customer is subscribed to certain services.
The user is authorised to use the service he/she is trying to access.

Use case description:

The user accesses the TES WWW site via a WWW-browser. On clicking on the link to access TES services the user is presented with a dialogue box in which he/she must enter a user ID and a PIN previously supplied by the TES provider. If the user enters an invalid user ID and/or PIN he/she is informed that their access has been denied. Otherwise the user is presented with a list of zero or more available services that he/she may access. The user may then select one of these services or opt to select none of them. If a service is selected further service specific interactions are then enabled. In addition the user will be presented with a way to exit the TES service that he/she has selected at any time.

5.2.2 Use Case: TES Customer Administrator sets up specific SAG intended for TES end users

Preconditions:
The customer already has an account with the TES provider.
The customer administrator is authorised to use the TES customer management service.
The customer is subscribed to the service the end user wishes to use
Use Case:
The TES Customer Administrator accesses the TES customer management service.
The Customer Administrator requests to create a new subscription assignment group, supplying the name of the group, the maximum size of the group and a description of the group. On successful completion of the operation a confirmatory message is provided to the customer administrator. The operation will fail if the name of the SAG already exists.

5.2.3 Use Case Breakdown

Once the overall functionality of the trial system had been defined by these use cases, the functionality had to be decomposed, both into that needed by systems operated by the different organisations in the trial and the functionality needed from the reusable components. The identification of which reusable components would be used in which organisational domains had largely be defined earlier in the project since the aim of the trial was partially to show this reuse of components. The detailed breakdown still needed to be recorded however and this was done in the context of each of the overall trial system use cases.

This breakdown was attempted using use cases and sequence diagrams. In the use case approach, the individual use cases were broken down into use cases related to individual systems and components and their relationship shown using UML use case diagrams. This therefore defined use cases for the individual organisations' systems, and used the use cases that had been defined for the components. Examples for the two uses cases given above as provided in Figure 5.4 and 5.5.

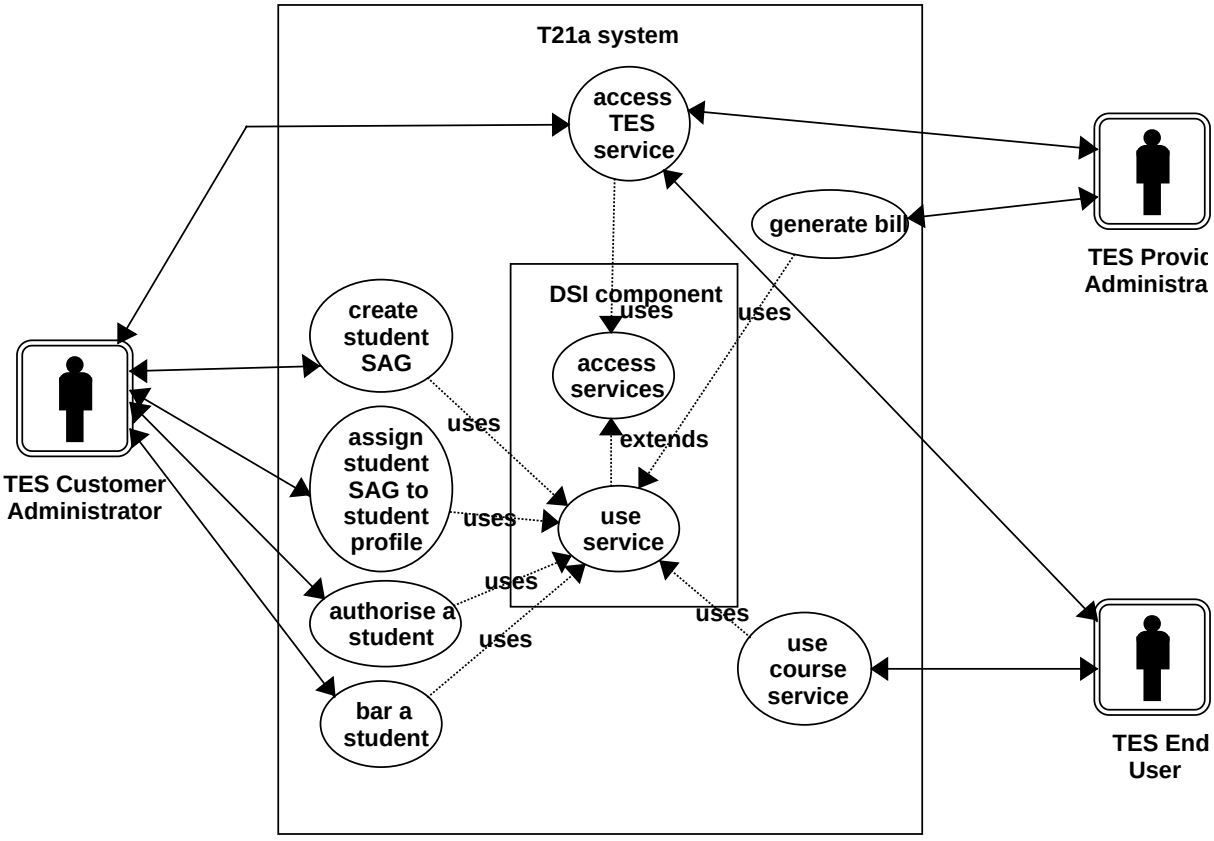


Figure 5.4: Breakdown of Access TES service use case

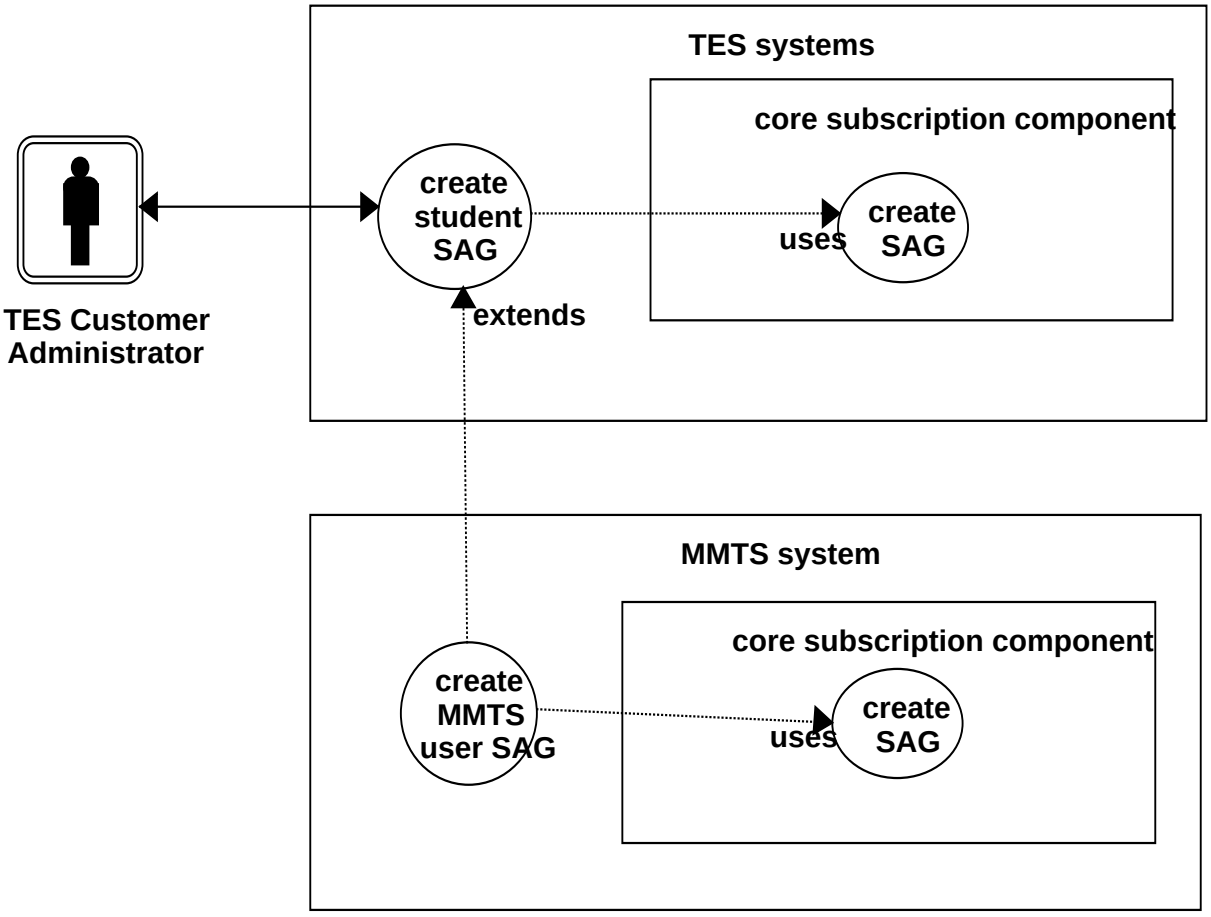


Figure 5.5: Breakdown of create student SAG use case

The sequence diagram approach used sequence diagrams that contained the various organisation systems and the components of which they were composed, and for specific use cases showed the resulting sequences of interactions between them.

Figure 5.6 below provides a summary of a subset of the primary use cases used in the Customer Managment Trial and how they map onto the different provider systems. (A single diagram consisting of all use cases in Customer Management Trial would be too complex to be of value).

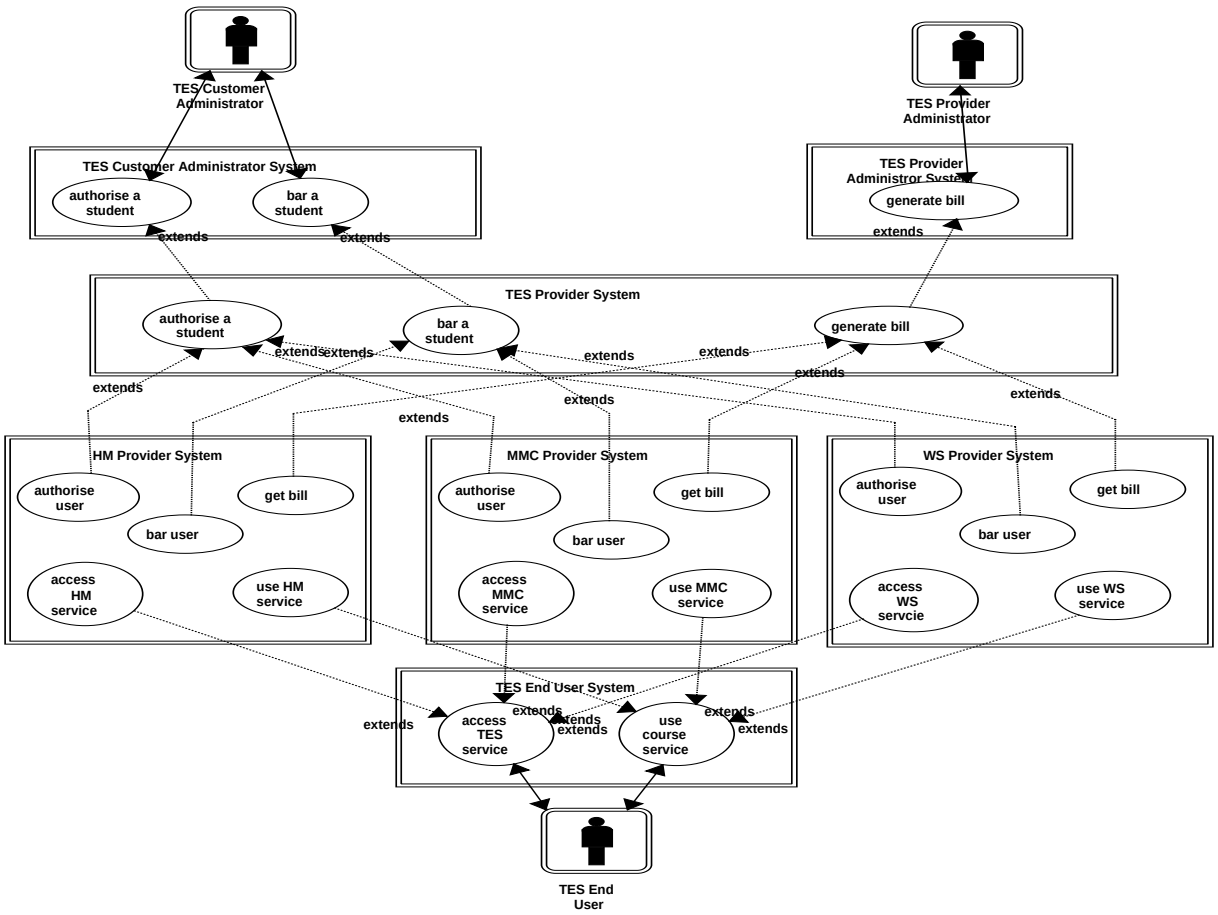


Figure 5.6: Customer Management Trial inter-domain use case summary

5.3 System Architecture

In the context of a multi-domain system in a commercial service provision situation, the multi-domain development process is unlikely to proceed to a stage involving detailed design, implementation and testing, this typically being left to the individual stakeholders involved. Instead, such a multi domain system development process is envisaged as a tool for use by those involved in arriving at industry agreements on the interfaces and process involved in open, inter domain management activities. The development process will principally be one of analysis and high level design on the multi-domain enterprise model and use case specification. Further refinement of the overall design occurs during the design of the different constituent domains or during the design of the reusable components. The ultimate benefit to industry of such analyses is common specification for software components that support management specific activities across several domains. The approach demonstrated in the previous section, where multi domain use cases are decomposed into use cases for specific components provides us with a means of identifying and specifying such components. The identification of reusable components, developed in Prospect for the systems involved in the Customer Management Trial, is therefore the main architectural result of the analysis process conducted at this multi domain level. This is represented in Figure 5.7 below.

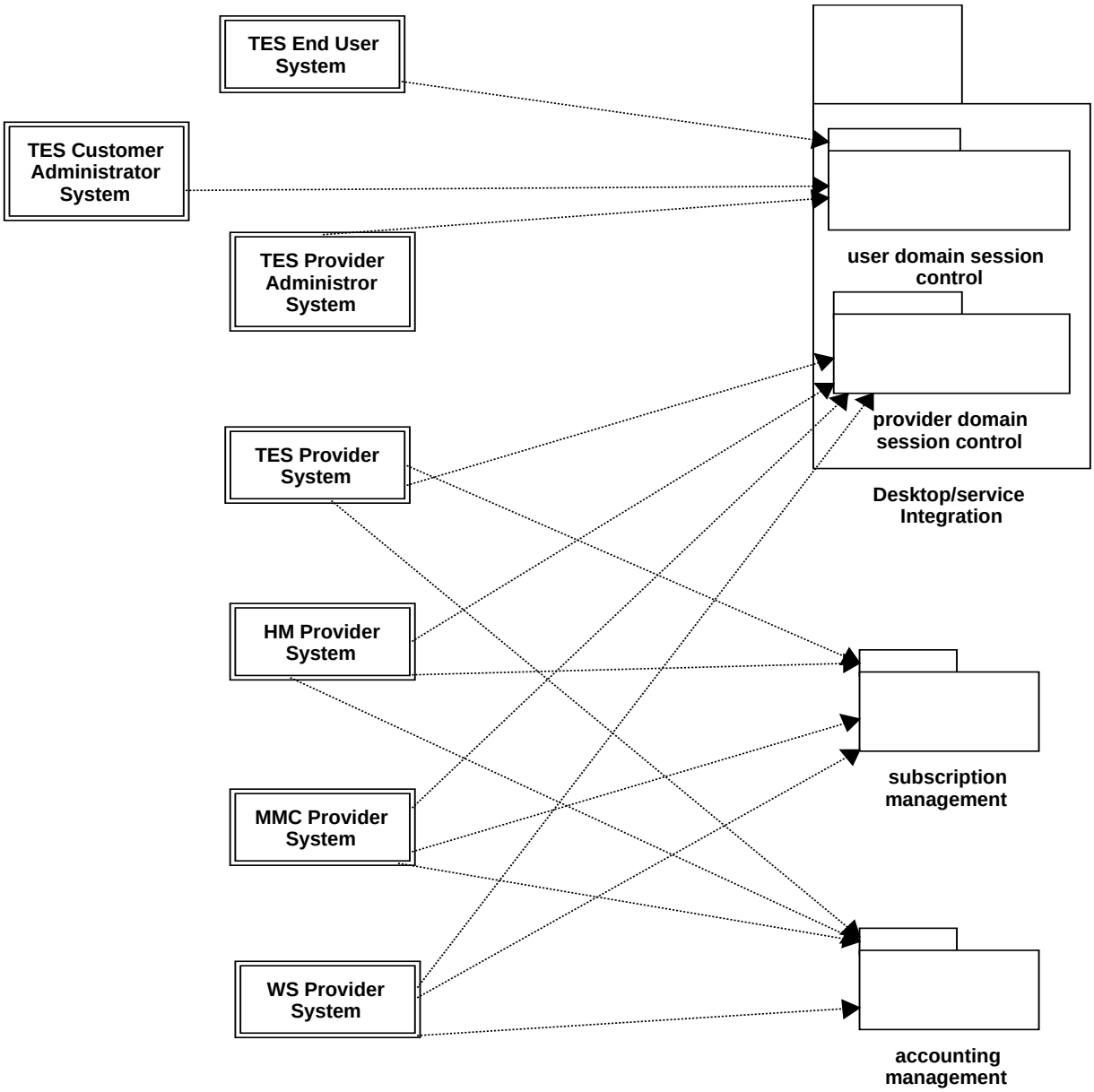


Figure 5.7: Use of reusable components in Customer Management Trial systems

6. Example Design of Inter Domain System using Prospect Design Process: Tele-education Service Management

The Prospect tele-education service provides educational courses to interested customers and is composed of a number of telecommunication services. Both information services like the Hypermedia Service and Communication Services like the Virtual Private Network Service (VPN) are integrated. The TES management system therefore has to co-operate with management systems of the subcontractors, across administrative boundaries. This is the major requirement imposed on the design of the management system.

Interworking between systems in different administrative domains is facilitated by re-use of software components in the systems. Therefore, implementations and specifications of the Prospect components were re-used where possible. As in Section 5, the design is presented as a detailed definition of the use cases and the object model.

6.1 Enterprise Model

The TES management system was developed based on the enterprise model depicted in Figure 6.1. The customer buys a tele-education service from the TES provider for students belonging to their organisation. Since this service is composed of several multimedia tele-services (TS), e.g. multimedia conferencing and WebStore, the TES provider in turn buys these services from the corresponding provider. All stakeholders support an administrator role, but only the TES customer and the TES provider administrator can use the TES management service.

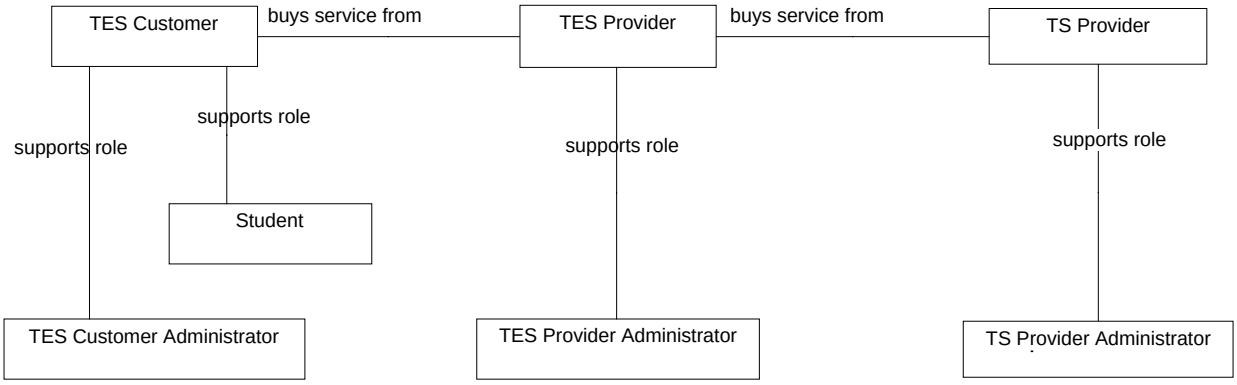


Figure 6.1: Enterprise Model - TES Management

6.2 Use Cases

Figure 6.2 shows the use cases defined for interaction with the TES management system. Two types of actors are supported; administrators of TES customers and of the TES provider. The TES provider administrator can act on behalf of a customer; i.e. all the use cases performed by the customer administrator can also be performed by the provider administrator.

The TES management use cases were derived from use cases defined for the subscription and the accounting component. The first of the scenarios described below was derived from a subscription use case, the second one from an accounting use case.

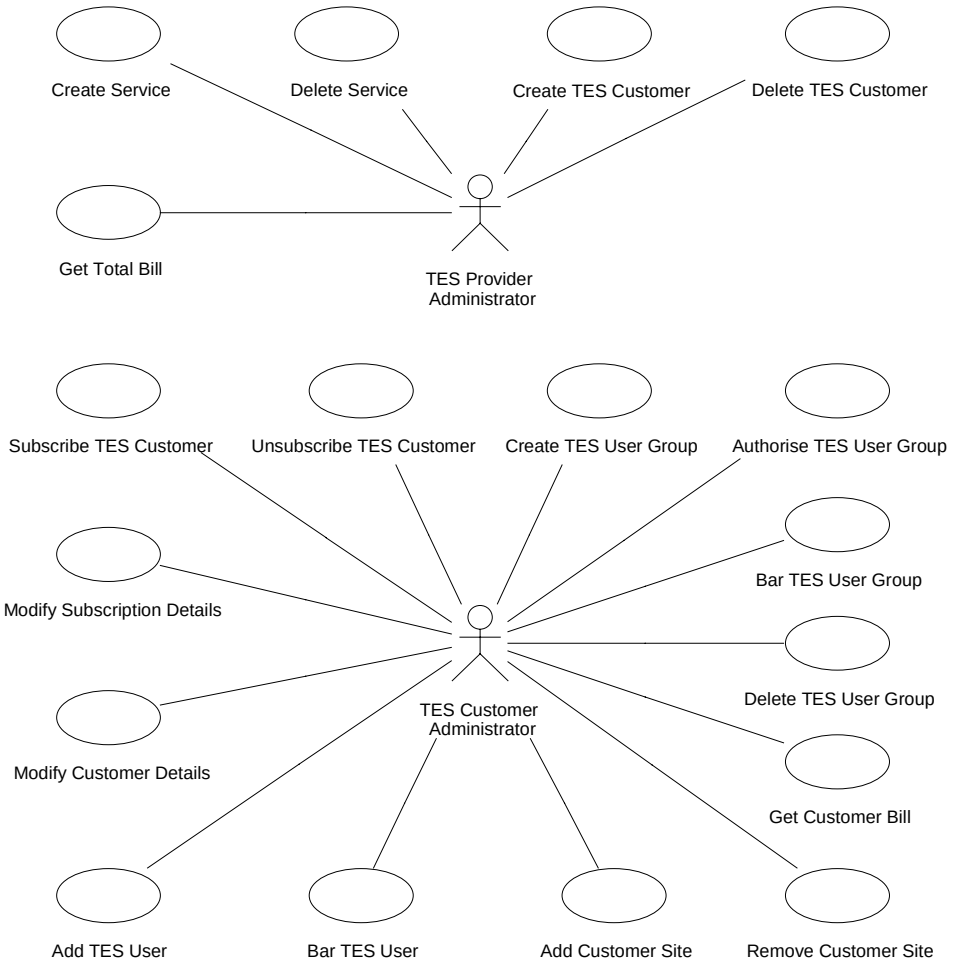


Figure 6.2: TES Management Use Cases

6.2.1 Subscribe TES Customer

Preconditions:

The TES management system is initialised. A service and the account of the customer have already been created. The administrator has access to a management application or service that is able to connect to the TES management system.

Use Case Description:

The administrator selects the service the customer wants to subscribe to. Further details of the subscription are defined, e.g. service profiles, tariffs.

Postconditions:

The customer is subscribed to the service and a subscription contract is concluded.

6.2.2 Get Customer Bill

Preconditions:

The TES management system is initialised. The customer is subscribed to a service. The administrator has access to a management application or service that is able to connect to the TES management system.

Use Case Description:

The administrator receives the bill for usage of the services the customer is subscribed to.

Postconditions:

The bill is displayed.

6.3 Object Model

The TES management system is composed of three adapted Prospect components; subscription, accounting, and session control (DSI). In Figure 6.3 the architecture of the management system is depicted.

Two different actors, namely the customer operator and the provider operator, can access the system by using a special management application that connects to the management service of the TES provider. That means, the TES management service is an extra service type available for all TES customers and for the TES provider.

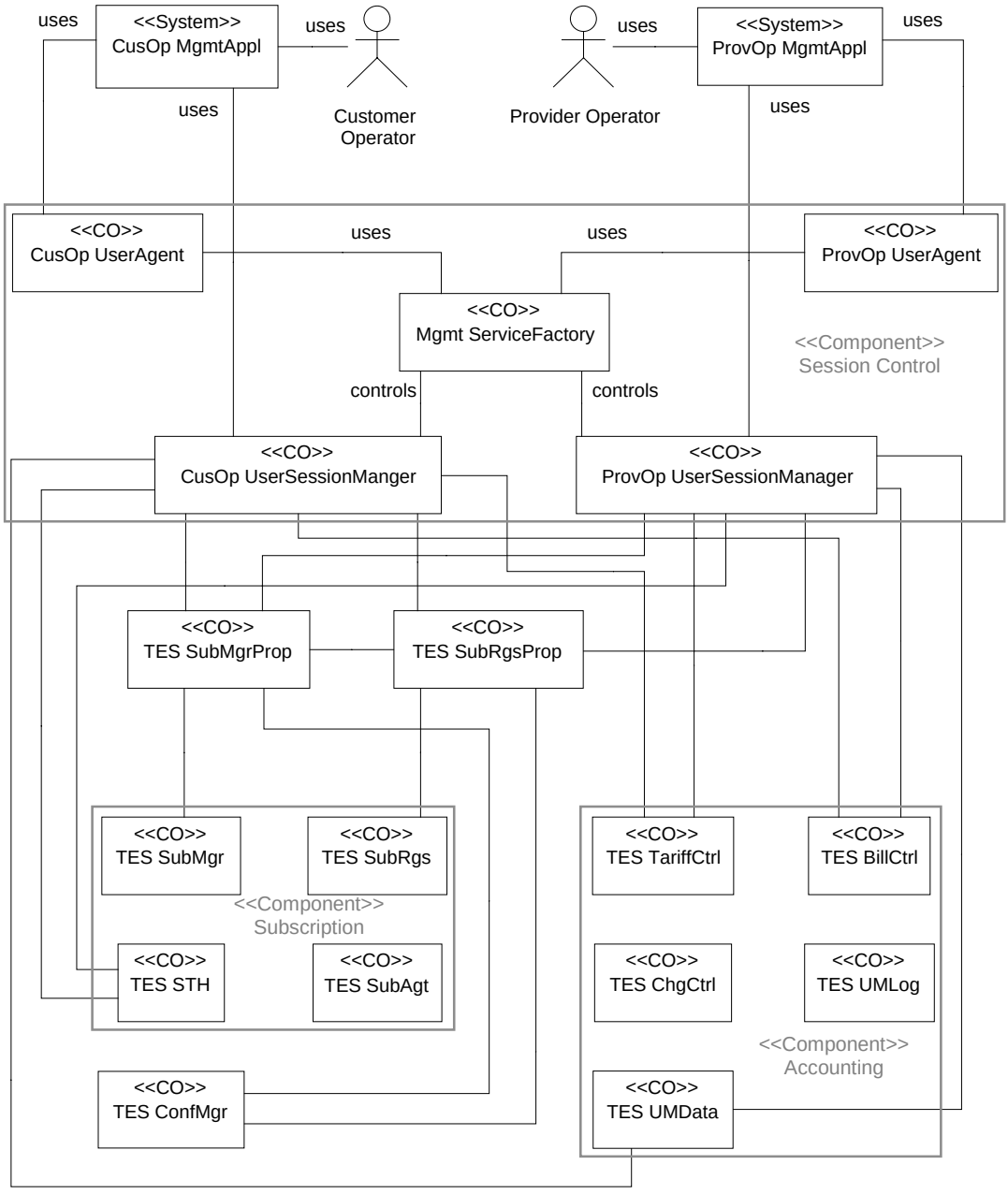


Figure 6.3: TES Management Design

Adaptation of the components was done in different ways. While only the DSI design and the accounting specifications were reused, an implementation of the subscription component was integrated with the system.

The overall DSI design and the specifications of user agent (UA) and service factory (SF) CO were reused. User session manager CO (USM) specifications for both customer and provider operators have been developed, based on the definition of the envisaged use cases. The specification of the accounting objects was reused, too. Propagation of accounting information from TES subcontractors is already supported by the component design.

Since it was planned to reuse an implementation of the subscription component, a number of additions to the component design were necessary. For instance, propagation of customer information to the TES subcontractors is not supported by the current implementation. Therefore, two TES specific COs have been specified in order to perform this task. The subscriber manager propagator CO (SMP) is responsible for forwarding the SAG management requests to the corresponding subscriber manager COs of the subcontractors. The subscription registrar propagator CO (SRP) forwards subscription requests to the subcontractors' subscription registrar COs. Both COs inherit interfaces from the corresponding COs of the subscription component, the subscriber manager and the subscription registrar CO.

Another TES specific CO is responsible for forwarding the requests to the VPN provider. The configuration manager CO (CM) is notified by the SRP whenever re-configuration of the network connections is needed, e.g. if a customer adds a new site to the list of access points available for their end-users.

In the following, two of the TES-specific COs are described briefly; the subscription registrar propagator CO and the customer management USM CO.

6.3.1 Subscription Registrar Propagator

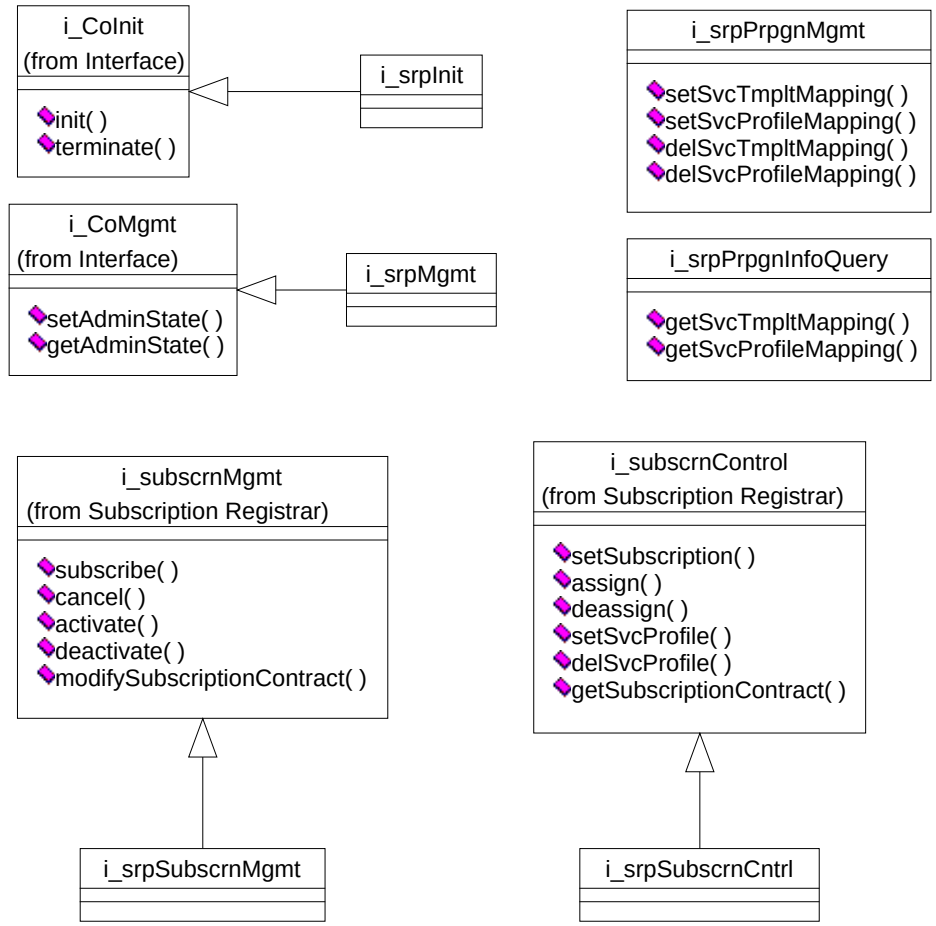


Figure 6.4 Subscription Registration

The subscription registration propagator CO (SRP) acts on behalf of the service provider in dealing with subscribers wishing to subscribe to a service. The SRP forwards requests to the local SubRgs CO and if necessary to SubRgs in subcontracted provider domains. It is responsible for the mapping of a service template to those used in the subcontractor domains and for the mapping of service profiles to those provided by its subcontractors.

The SRP CO maintains the following information:

- the mapping of service templates.
- the mapping of service profiles.

One SRP operates in the service provider domain. The CO provides the following interfaces:

- *Object Lifecycle Management Interface* (i_srpInit): This interface provides operations for initialising and terminating this object.
- *Object Management Interface* (i_srpMgmt): This interface provides operations for controlling the administrative state of the CO.
- *Subscription Management Interface* (i_srpSubscrnMgmt): This interface provides operations for set-up and modification of service subscriptions. It inherits from the interface i_subscrnMgmt of the SubRgs.
- *Subscription Control Interface* (i_srpSubscrnCtrl): This interface provides operations for set-up and modification of service profiles. It inherits from the interface i_subscrnControl of the SubRgs.
- *Propagation Management Interface* (i_srpPrpgnMgmt): This interface allows the provider operator to manage service profile mappings.
- *Propagation Information Query Interface* (i_srpPrpgnInfoQuery): This interface provides the provider operator with information about service template mappings and service profile mappings.

6.3.2 Customer Operator User Session Manager

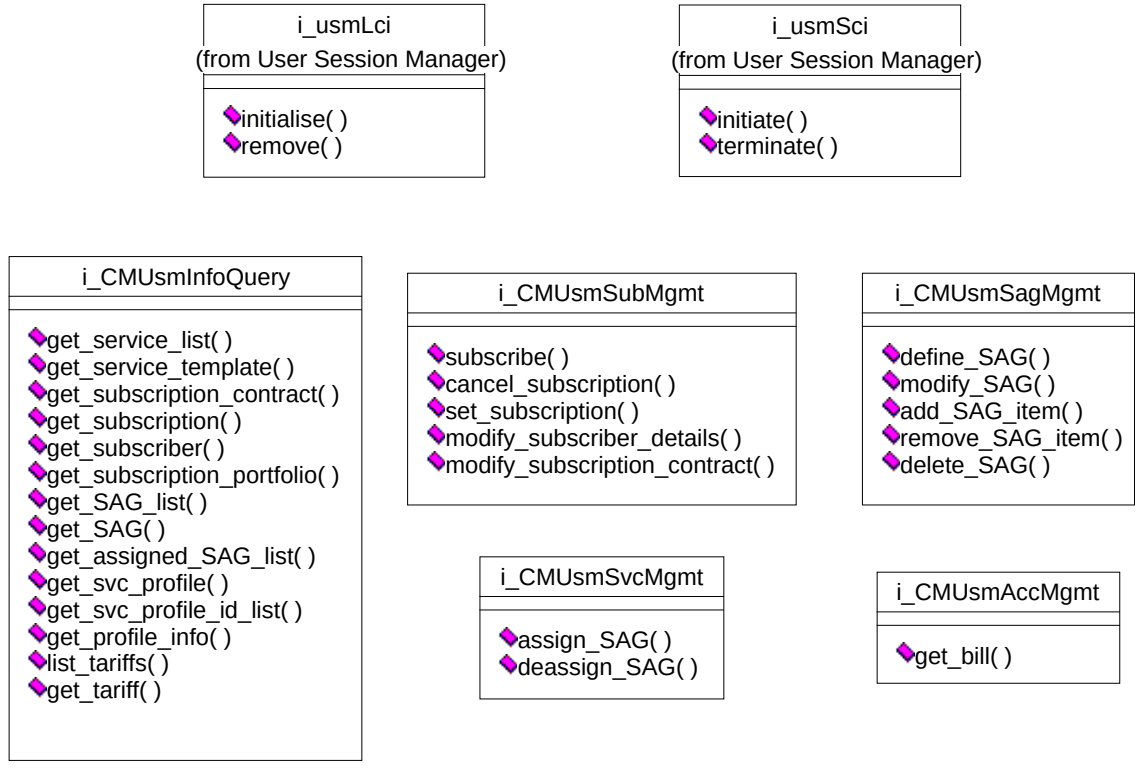


Figure 6.5 User Session Manager

The customer management USM (CMUSM) consists of a number of interfaces which provide the necessary management functionality for TES customers, e.g. subscription and SAG management. The CMUSM is (TES) service independent and one operates per customer operator.

The CMUSM CO provides the following interfaces:

- *Life-cycle Management Interface* (i_usmLci): This interface provides operations for initialisation and deletion of the object.
- *Session Control Interface* (i_usmSci): This interface provides operations for initiating and terminating end-user sessions.
- *Information Query Interface* (i_CMUsmInfoQuery): This interface provides the customer operator with information about services, subscriptions, SAGs, and profiles.
- *Subscription Management Interface* (i_CMUsmSubMgmt): This interface allows the customer to subscribe to services.
- *Subscription Assignment Group Management Interface* (i_CMUsmSagMgmt): This interface allows the customer operator to manage user groups.
- *Service Management Interface* (i_CMUsmSvcMgmt): This interface provides operations to manage the assignment of user groups to service profiles.
- *Accounting Management Interface* (i_CMUsmAccMgmt): This interface provides operations to get accounting details.

6.4 Sequence Diagrams

In the following subsections sequence diagrams are presented that illustrate the object interactions for the use cases defined in Section 6.2.

6.4.1 Subscribe TES Customer

Preconditions:

The TES management system is initialised. The customer and the service IOs are initialised. Either the customer or the provider operator have initialised the management application and have access to the TES management service.

Sequence:

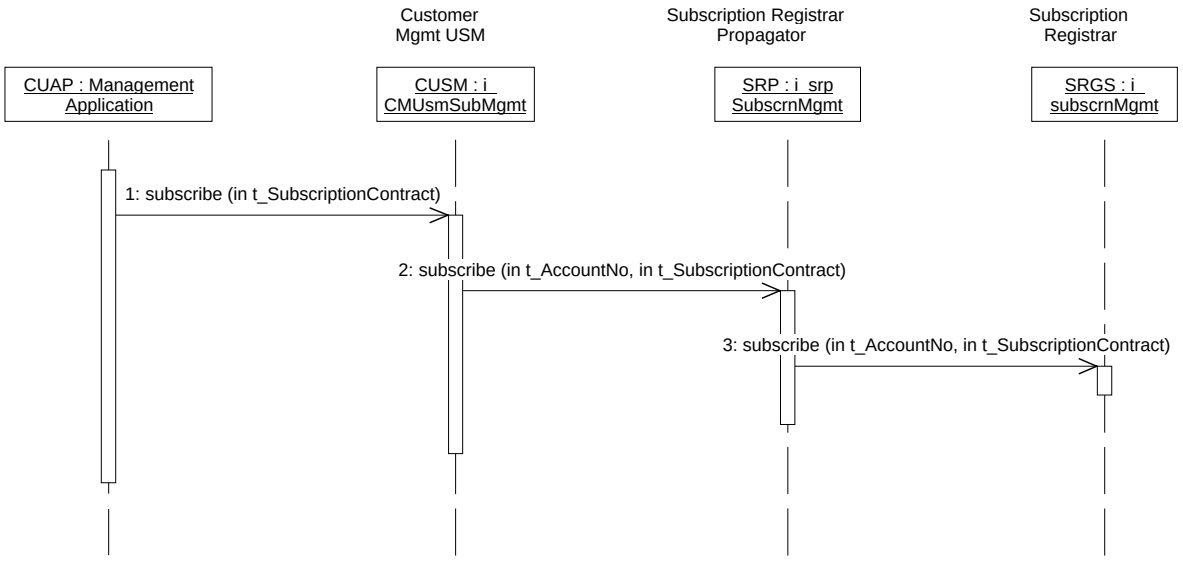


Figure 6.6 Subscribe TES Customer

1. The customer operator selects the service he wants to subscribe to and invokes the subscribe operation at the USM.
2. The USM forwards the request to the subscription registrar propagator CO.
3. The SRP forwards the request to the local subscription registrar CO.

6.4.2 Get Customer Bill

Preconditions:

The TES management system is initialised. The customer is subscribed to the service. Either the customer or the provider operator have initialised the management application and have access to the TES management service.

Sequence:

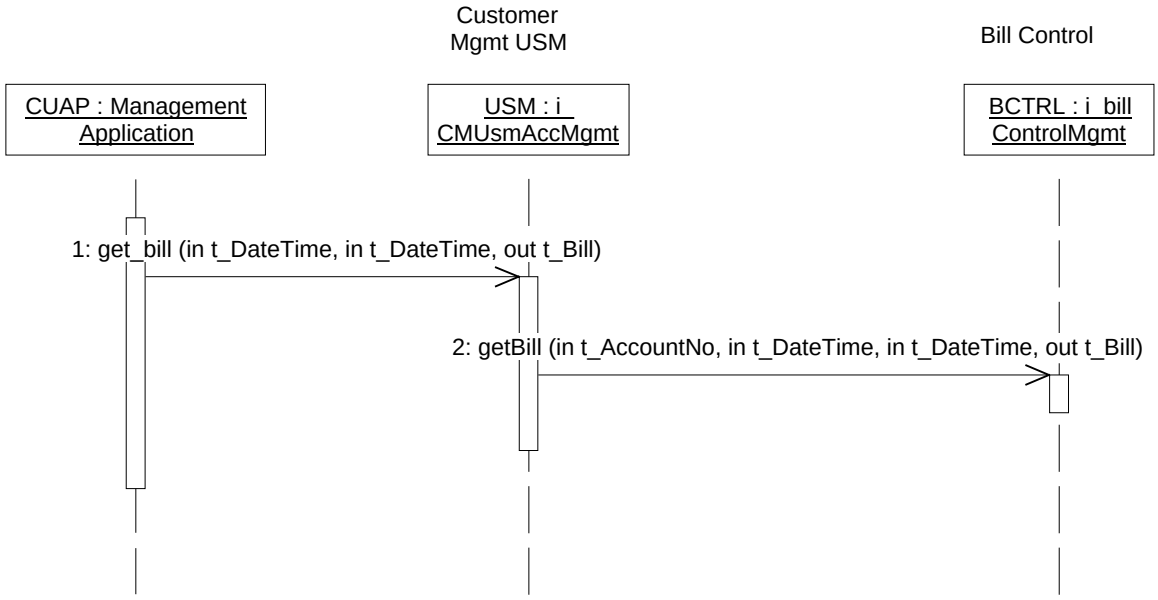


Figure 6.7 Get Customer Bill

1. The operator invokes the get_bill operation at the USM.
2. The USM forwards the request to the local bill control CO.

7. Example of Design of Re-usable Component using Prospect Design Process: Subscription Management

This section presents the use case and IDL definitions for the subscription management component of Prospect. It is a primary aim of this design that a common IDL specification can be applied to the implementations of all the stakeholder systems (i.e. within each Prospect domain) exercising subscription management and that consequently common implementations can be used where possible. This would facilitate design and specification and where possible software reuse.

The design is presented as a detailed definition of the use cases and the object model. This provides sufficient basis for implementation of components that interact with the subscription management components as well as for the implementation of the subscription management components themselves.

7.1 Enterprise Model

The subscription component was designed based on the simple business model of a customer and a telecommunication service provider having a contractual relationship with each other (see Figure 7.1). Usually, the component is located in the provider's administrative domain.

Both stakeholders support an administrator role. The provider administrator operates the telecommunication services; i.e. configures and maintains the offered services. The customer administrator is responsible for provision of all information the provider requests, e.g. the identity of the customer's end-users.

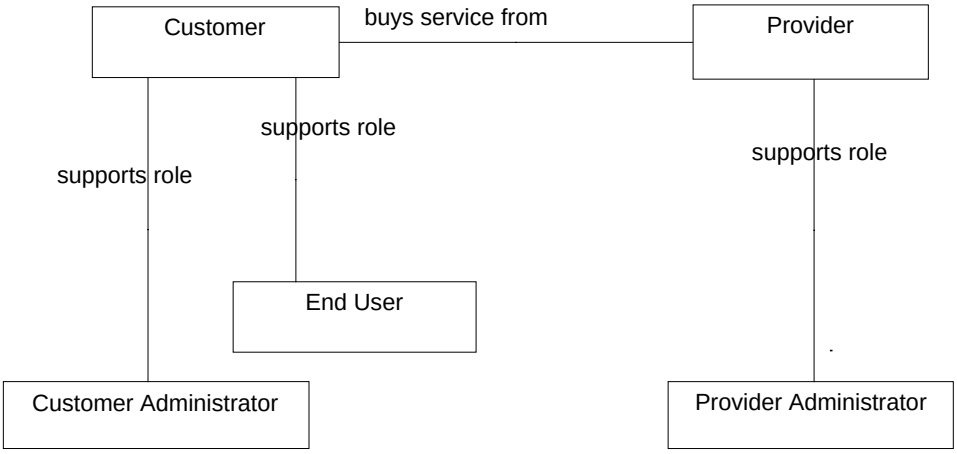


Figure 7.1: Enterprise (Class) Model - Subscription

7.2 Use Cases

In Figure 7.2 the use cases for interaction with the subscription component are depicted using UML notation. The component is merely used by the administrators of the service provider and service customer. The subscription component interacts with the DSI component, which is used by end-users to get access to the service.

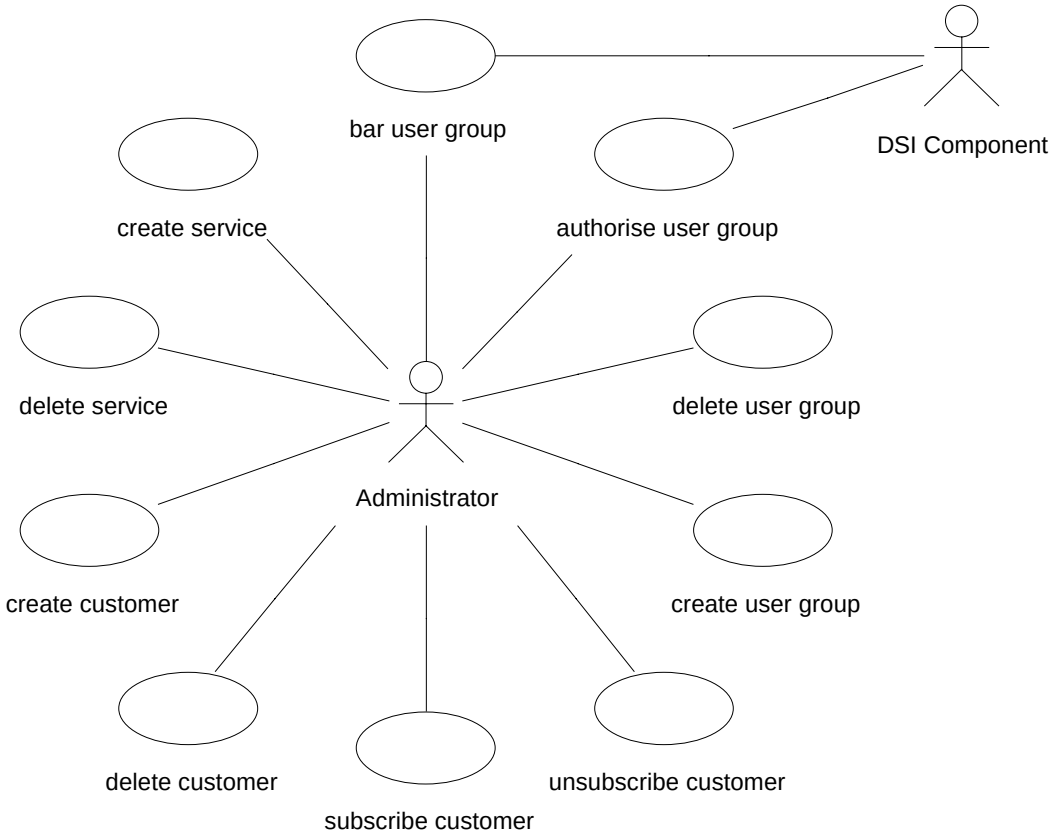


Figure 7.2: Subscription Use Cases

The following two subsections illustrate the definition of use cases in plain text. In the first use case, a customer is subscribed to a service. In the second one, the customer authorises a group of end-users to access a subscribed service.

7.2.1 Subscribe Customer

Preconditions:

The subscription component is initialised. The service and an account for the customer have already been created. The administrator has access to a management application, which is able to connect to the subscription component.

Use case description:

The administrator selects the service the customer wants to subscribe to. Further details of the subscription are defined, e.g. service profiles, tariffs.

Postconditions:

The customer is subscribed to the service and a subscription contract is concluded.

7.2.2 Authorise User Group

Preconditions:

The subscription component is initialised. The service and an account for the customer have already been created. The customer is subscribed to that service and a user group has been created. The administrator has access to a management application, which is able to connect to the subscription component.

Use case description:

The administrator assigns the user group to the service subscription.

Postconditions:

The end-users of the user group are able to access and use the service.

7.3 Object Model

The subscription component should support for subscribers to allow end-users to access services. The subscription COs therefore define generic capabilities needed to manage, from the subscription point of view, different types of services. The COs related to subscription include (see Figure 7.3):

- Service template handler,
- Subscriber manager,
- Subscription registrar, and
- Subscription agent.

The service template handler is responsible for handling service templates that represent service capabilities provided by the provider. It also provides operations, such as verification of subscription parameters, to the corresponding subscription registrar. The subscriber manager manages information associated to subscribers, e.g. subscriber details and user groups.

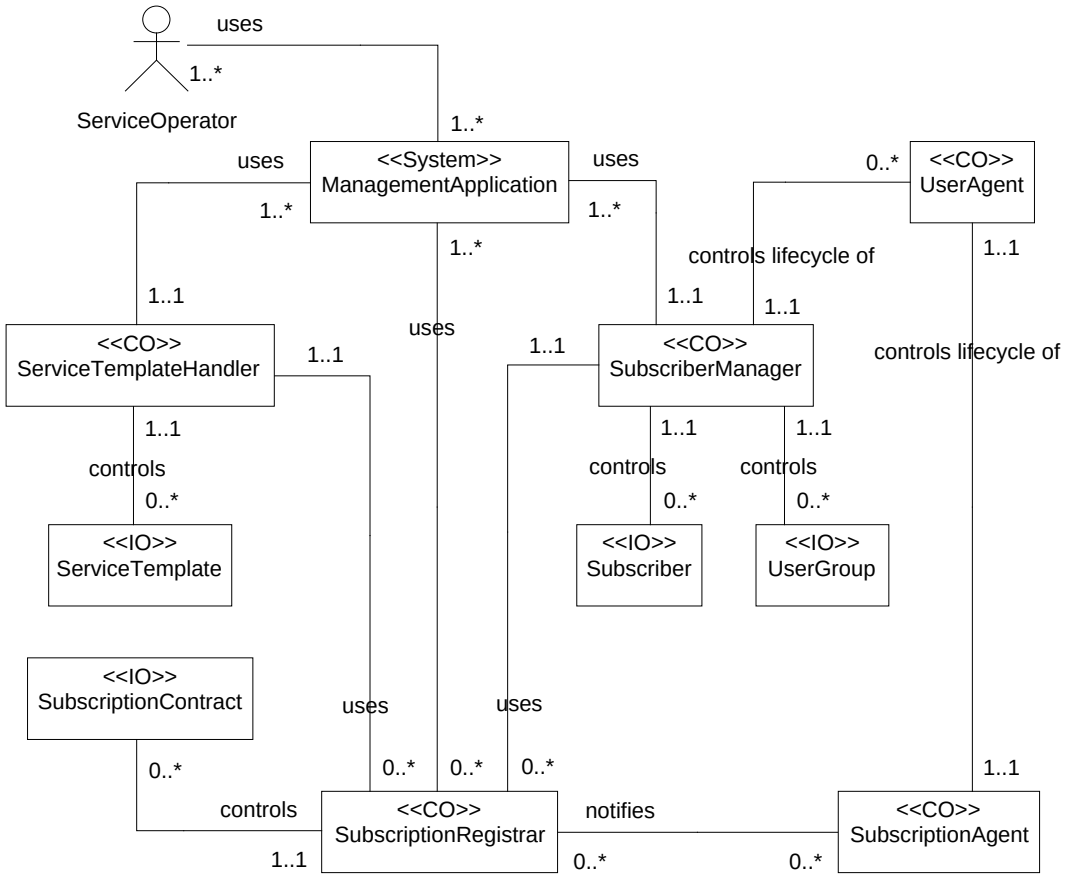


Figure 7.3: Design of the Subscription Component

The subscription registrar acts on behalf of a provider in dealing with subscribers wishing to subscribe to a service. A single subscription registrar can collect information for one service from one or more subscribers and will handle exactly one service, since each service is likely to have different requirements

for subscribing a customer to a service. A subscription registrar is responsible for managing of subscription contracts, among others.

The subscription agent is closely related to the user agent CO that belongs to the DSI component. It sends appropriate subscription information to the user agent, in reply to its request. The subscription component is accessed by a management application. Separate applications may be developed for customer and provider administrators.

In the following two subsections the specification of the subscription contract IO and the interfaces of the subscription registrar CO are presented. The interfaces should correspond with those used for the CORBA interfaces making up the corresponding engineering object.

7.3.1 Subscription Contract

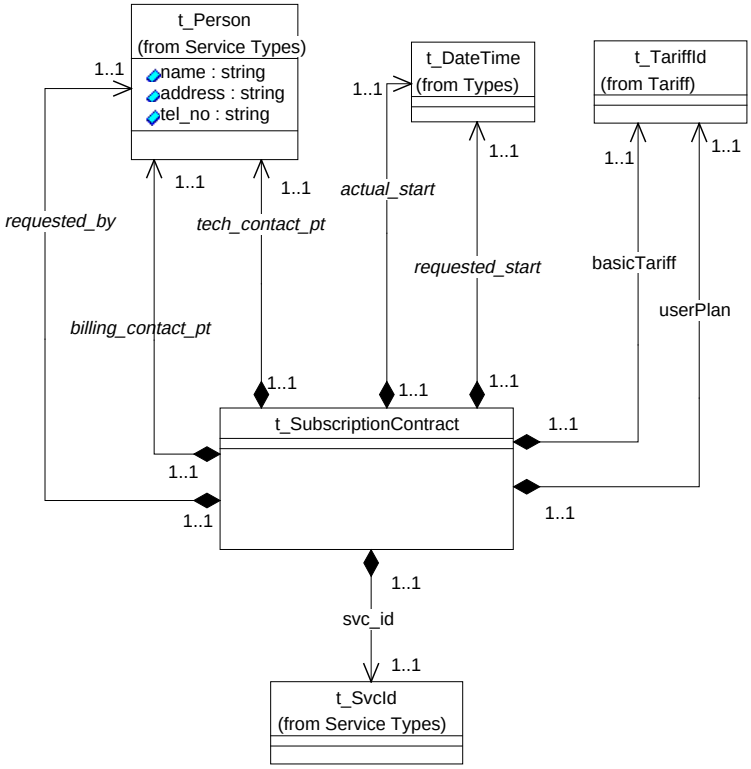


Figure 7.4 Subscription Contract

A subscription contract comprises all details of the subscriber to provider relation. The IO t_SubscriptionContract contains the following information:

- *Service Identifier* (t_SvcId): This is an unique identifier for the service.
- *Requestor* (t_Person): This is the person in the customer domain who requested the subscription contract.
- *Billing Contact* (t_Person): This is the person in the customer domain responsible for payment of the bill.
- *Technical Contact* (t_Person): This is the technical contact person in the customer domain.
- *Requested Start* (t_DateTime): This is the requested start date and time.
- *Actual Start* (t_DateTime): This is the actual start date and time.
- *Basic Tariff* (t_TariffId): This is an unique identifier for the basic tariff the subscriber has to pay.
- *User Plan* (t_TariffId): This is a unique identifier for a special tariff for a subscriber, e.g. a discount.

7.3.2 Subscription Registrar

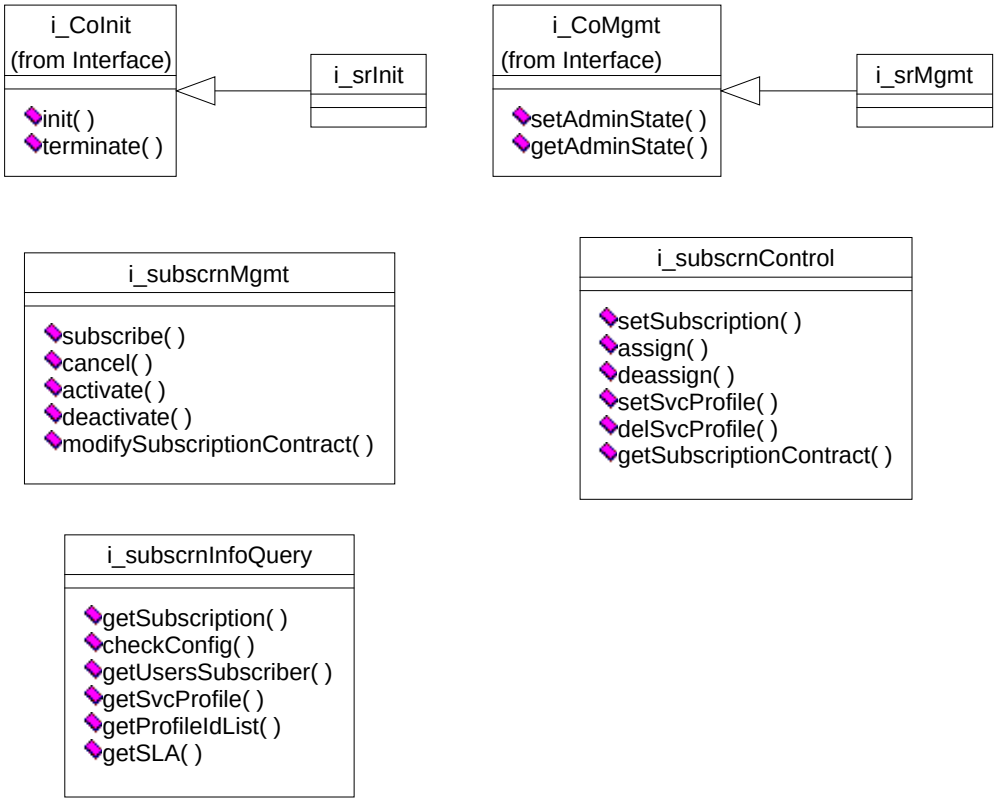


Figure 7.5 Subscription Registrar

The subscription registrar (SubRgs) CO acts on behalf of the service provider in dealing with subscribers wishing to subscribe to a service. The subscription registrar activates subscriptions to a service, collecting subscription information from administrators and logging/modifying subscription information. A single SubRgs is responsible for exactly one service, since each service has different requirements for subscription and authorisation. The CO also authorises subscription assignment groups.

The SubRgs is service dependent. One SubRgs exists in each service provider domain for each separate service (as defined by a service template) that is offered by that provider. The behaviour of the SubRgs may vary depending on the operational semantics of the service it supports.

The subscription registrar CO maintains the following information:

- Subscription contracts
- Subscriptions, including profiling information affecting all subscription assignment group IOs (SAG) operating under this subscription
- Service profiles
- SAGs that have service profiles assigned to them
- The relationships between service profiles and SAGs for different subscriptions.

The Subscription Registrar CO provides the following interfaces:

- *Object Lifecycle Management Interface* (i_srInit): This provides operations for initialising and terminating this object.
- *Object Management Interface* (i_srMgmt): This provides operations for controlling the administrative state of the CO.
- *Subscription Management Interface* (i_subscrnMgmt): This interface allows for the set-up and subsequent modification of service subscriptions.

- *Subscription Control Interface* (i_subscrnControl): This provides operations for controlling the subscription of customers to the service by manipulating subscription contracts and service profiles.
- *Subscription Information Query Interface* (i_subscrnInfoQuery): This provides operations for obtaining information about subscription contracts and service profiles. This interface is available to the service customer.

7.4 Sequence Diagrams

The sequence diagrams presented below illustrate the object interactions for the use cases introduced in Section 7.2.

7.4.1 Subscribe Customer

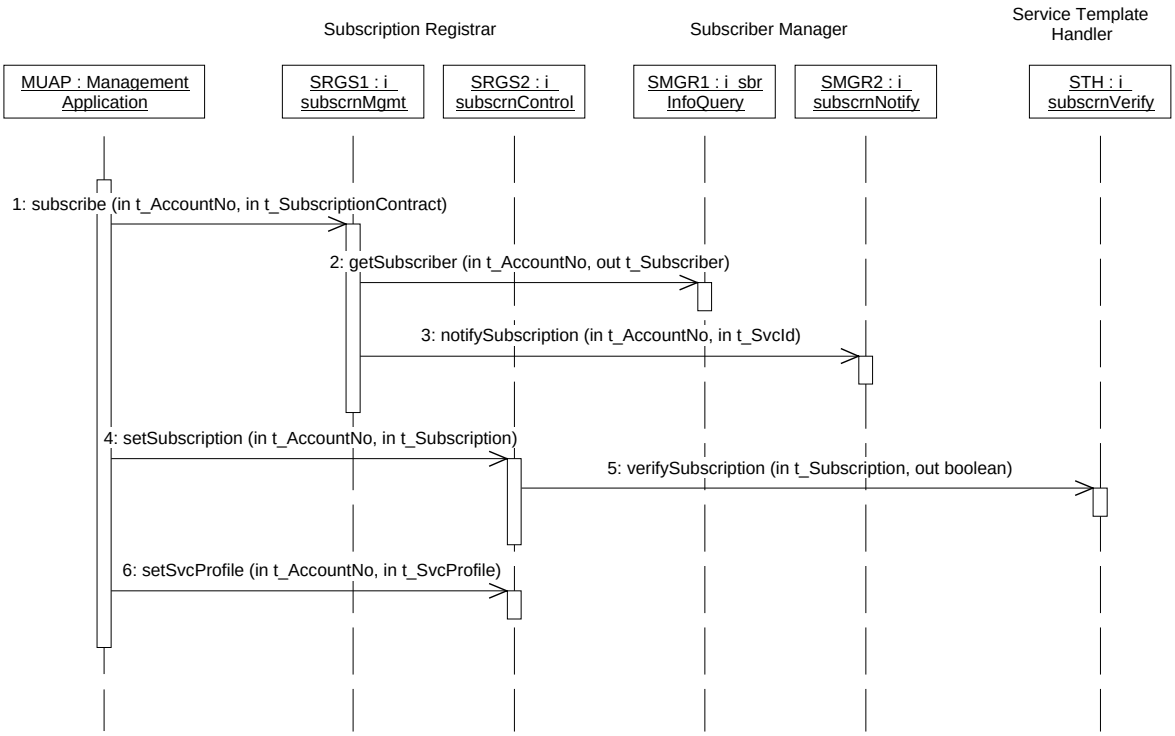


Figure 7.6 Subscribe Customer

Preconditions:

1. Use cases ‘Create Service’ and ‘Create Customer’ were successfully executed.

Sequence:

1. The subscription contract for the subscriber is initiated. The SubRgs receive an account number, the service id, and other service details.
2. The SubRgs checks with the SubMgr that there is a subscriber with such an account number.
3. The subscription contract is registered by the SubRgs in the subscription portfolio held for the subscriber in the SubMgr. If a portfolio does not already exist this action causes one to be instantiated.
4. The SubRgs registers the subscription to the service.
5. The SubRgs requests the STH to verify that the subscription details are within the parameters allowed for this service as specified in the service template.
6. The service profile is set for the subscription.

7.4.2 Authorise User Group

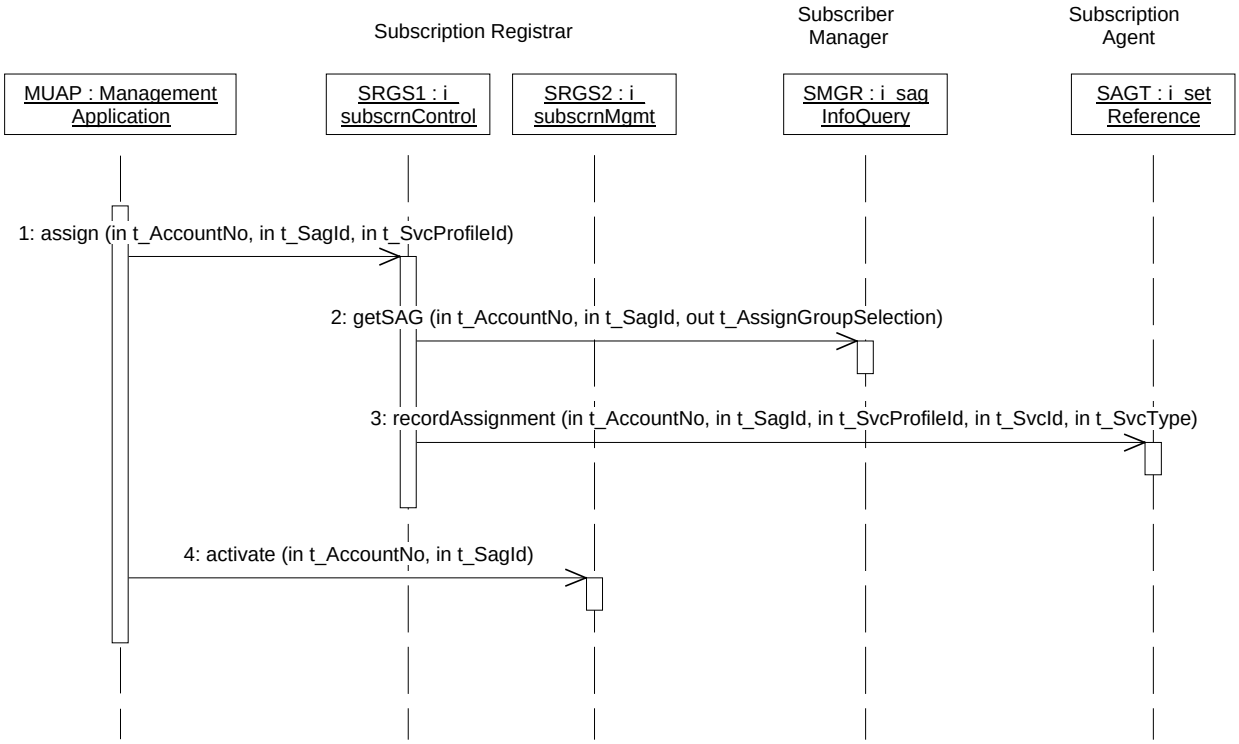


Figure 7.7 Authorise User Group

Preconditions:

1. Use cases ‘Create User Group’ and ‘Subscribe Customer’ were successfully executed.

Sequence:

1. The service profile is assigned to the SAG.
2. The SubRgs receives the SAG details from the SubMgr.
3. Each SubAgt is notified about the assignment of the service profile to the SAG. This enables these users to now access the service within the constraints of the service profile.
4. The SubRgs activates the SAG which involves setting the corresponding service profile to the activate state.

8. Conclusions

8.1 Choice of Notation Language and Specification techniques

UML can be thought of as a 'family' of modelling notations. However, Prospect had to choose a subset of these notations which were most relevant to the system under construction. Prospect chose Use cases, Object Model, Sequence Diagrams. These provide the necessary expressiveness for describing the enterprise, object structure and relationships, and object interaction models required in the design and development of the Prospect trial system.

The use cases and sequence diagrams were very useful in specifying the necessary testing and integration stages of the design process. They also provided a very useful way of documenting the design and implementation in a form relatively easy to understand. IDL was used to specify the component and systems interfaces.

Several ancillary issues not discussed in detail in this document, but which had an effect on the interfaces developed, were the naming conventions used within the project and the choice of the CORBA Common Object Services used in the trial (e.g. naming services). These issues are more appropriately discussed in the Prospect Trial 2 System description [D4b-98].

8.2 Tool support for UML

Prospect made use of two modelling tools which supported UML: Paradigm Plus from Platinium and ROSE from Rational. Both of these tools supported UML v1.0 notations for Use Case specification, Object Model Representation, and Sequence Diagrams. These tools proved very effecting the design and modelling stages of the implementation. Some IDL code which was generated by the tools, was used in the trial implementation. Overall the experience in developing systems using both tools was quite favourable.

The tools were also very useful in generating documentation for the trial system implementations. The trial system design would have been too time consuming to have been developed completely by hand, and the use of the tools provided a more unified interpretation and usage of the modelling notations. Their usage also aided in maintaining the consistency of the modelling and specification effort.

Although the tools do not support the complete set of UML notations, the basic diagrams and stereotypes are available in both tools. Besides the use case diagrams used in the analysis phase, class diagrams and sequence diagrams have been extensively used in the system specification. Classes and the associations between them, e.g. aggregation and generalisation, can be modelled in detail as well as sequences of interactions between the class instances.

On the other hand, additional support for some important UML notations would have made specification of the CORBA systems much easier. In Rational Rose, for instance, the interface sterotype is missing. Therefore, interfaces had to be modelled as classes and could not easily be distinguished from the information objects. Both tools do also not support for inclusion of multi-interface objects in sequence diagrams which is useful to show interactions between computational objects. In order to do this, new composed objects had to be specified. It is hoped that, as the UML standard evolves, future versions of both Rational Rose and Paradigm Plus support all the notations needed for modelling of CORBA systems.

It was not possible to standardise on a single vendor's tool within the corsortium. This caused some difficulty as it was not possible to transfer UML models from one tool to another. The interchange of models between CASE tools is a recognised problem and a standard for model exchange is being agreed but is not yet supported by the products.

8.3 Conclusion on Design Process

The design process provided a well structured, architecturally driven methodology for developing end-to-end multi domain management solutions. The design process proved effective in co-ordinating the design and implmentation effort. It facilitated the re-use of existing (TINA) component designs as well as the specification of re-usable components (i.e. implementations of the accounting and subscription components were reused in several provider domains). It adapted the use of UML and IDL notations and specifications languages to the problems of multi-domain management systems development. The design process has also been adopted as a basis for the ACTS Guideline on Designing Service Management System (a concensus document indicating acceptance and agreement between several network and service management projects in the ACTS programme) [Wade-97]. Earlier versions of the design process has also been published at international conferences and workshops [Wade-97a].

8.4 Possible future enhancements for the Prospect Design Process

Within the limited confines of a trial based project such as Prospect, it was not possible to realise the specification of design patterns for the components and sub components. However, the designs and specification developed in Prospect can serve as a basis on which to develop generic design patterns. Candidate design patterns would possibly include areas of accounting, subscription and certain types of performance management. Further extension of the design process would be needed to specify the templates and identify the modelling notations which would constitute design pattern descriptions. However, it is anticipated that the choice of existing modeling notations used in the design process would be adequate for this task.

Glossary of Definitions used in Prospect Design Process

Constraint - is a semantic relationship among model elements that specifies conditions and propositions that must be maintained as true (otherwise the system described by the model is invalid). A constraint is shown as a text string in { } brackets.

Package - is a group of model elements. Can be nested. Is used to 'modularise' model description.
A package is depicted as a large rectangle with a small rectangle (called a 'tab') attached on one corner.

Type-Instance correspondence - the type of model element and an instance of that model instance use the same graphical representation. To differentiate between the type and an instance of a type of an model element by underlying the name of the instance.

UML stereo Types - is a new class of modelling element. It must be based on (derived from) existing classes in the UML meta model. represents a built in way of extending UML.

Class Diagram - is a graph of modelling elements shown on a two dimensional surface.
They are organised into packages. (This is probably better called a Static Structural Diagram as it can include types, packages, relationships and instances however 'Class Diagram' is UML's preferred name for it).

Object Diagram - is a graph of instances.
A static object diagram is an instance of a class diagram.
A dynamic object diagram shows the detailed state of a system over some period of time (including the changes that occur in that period of time).
Dynamic object diagrams are manifested as collaboration diagrams.
Note: There is no separate format for an object diagram. A class diagram can contain objects so a class diagram with objects and no classes is an 'object diagram.

Class - is a descriptor for a set of objects with similar structure, behaviour and relationships.
UML provides a graphical notation for declaring and using classes as well as a textual notation for referencing classes within the description of other model elements.
Note: A class shown within a package can be referenced using the syntax PackageName::Class-name

List compartment - holds a list of strings each of which is the encoded representation of an element e.g. attribute, operation.
In addition lists can show other kinds of pre-defined or user-defined values such as responsibilities, rules, or modification histories.

Type - A type is a descriptor for objects with abstract state, concrete external operation specification and no operational implementations.

Interfaces - An interface is the use of a type to describe externally-visible behaviour of a class, component or other entity (e.g. packages).
They can be represented (graphically) as a rectangle symbol (with compartments) or in shorthand using a small round circle (usually with a solid line connecting it with the class which implements the interface and a label naming the interface).

Utility - is a grouping of global variables and procedures in the form of a class declaration. (It is modelled as a stereotype off a class).

Metaclass - is a class whose instances are classes.

Importing a package - a class in another package may be referenced. This is captured by an 'imports dependency' relationship between the two packages.

Operations - an operation is shown as a text string that can be parsed into various properties of an operation model element. Has default syntax

```
visibility name ( parameter-list): return-type-   expression { property-string }
```

+ visible
# protected,
- private

Association - binary associations are shown as lines connecting class symbols. The lines may have a variety of adornments to show their properties. Ternary and high order associations are shown as diamonds connected to class symbols by lines.

Binary association is drawn as a solid path connecting two class symbols.

Note: An 'or association' indicates a situation in which only one of several potential associations may be instantiated at one time for any single object. This is shown as a dashed line connecting two or more associations, all of which must have a class in common with the constraint string 'or' which labels the dashed line. Any instance of the class may only participate in at most one of the associations at one time.

There are also Association Role, Association Class. Qualified Associations, N-ary Associations

Composition - is a form of aggregation with strong ownership and coincident lifetime of part with the whole. Represented as a filled in diamond between the 'owner' element and the 'owned' elements.

Generalisation - is the taxonomic relationship between a more general element and a more specific element that is fully consistent with the first element and that adds additional information.

It is used for classes, packages, use cases and other elements.

It is drawn as a solid line from the more specific element (e.g. sub-class) to the more general element (e.g. super-class) with a large hollow triangle at the end of the end of the path where it meets the more general element.

Dependency - a dependency indicates a semantic relationship between two (or more) model elements.

It relates the model elements themselves and does no require a set of instances for its meaning.

It indicates a situation in which a change to the target element may require a change to the source element in the dependency.

It is shown as a dashed arrow from one model element to another model element that the first model element is dependent upon.

Use Case - A use case diagram is a graph of actors, a set of use cases enclosed by a system boundary, communication (participation) associations between the actors and the use cases, and generalisations among the use cases.

It is drawn as an ellipse containing the name of the uue case.

Actor - an actor is shown as a class rectangle with the stereotype "actor". The standard stereotype icon is a 'stick man' figure with the name of the actor below the figure.

Use Case Relationship - There are several meaningful relationships within a use case. These are

Communicates where the participation of an actor in a use case is shown by connecting the actor symbol by a solid path. The actor is said to communicate with the use case.

Extends - am extends relationship between use cases is shown by a generalisation arrow from the use case providing the extension to the base use class.
Uses - A 'uses' relationship between use cases is shown by a generalisation arrow from the use case doing the sue to the sue case being used.

Interaction diagrams - show a pattern of interaction among objects. Interaction diagrams come in two forms based on the same underlying information but emphasising a particular aspect of it (sequence diagrams & collaboration diagrams)

Sequence Diagrams - a sequence diagram has two dimensions: the vertical dimension represents time, the horizontal dimension represents different objects. Normally time proceeds down the page.

Object Lifeline - an object is shown as a vertical dashed line called a life-line. The life-line represents the existence of the object at a particular time.
If the object is created or destroyed during the period of time shown on the diagram, then its lifeline starts or stops at the appropriate point - otherwise it goes from the top to the bottom of the diagram.

Message - a message is a communication between objects that conveys information with the expectation that action will ensue. The receipt of a message is normally considered an event.

Collaboration diagram - shows an interaction organised around the objects in the interaction and their links to each other. A collaboration diagram shows the relationships among objects but does not show time as a separate dimension. Therefore the sequence of messages and the concurrent threads must be determined using sequence numbers. It only represents the objects and messages involved in accomplishing a purpose or a related set of purposes, projected from the larger system of which they are part.

Collaboration - is a modelling unit that describes a set of interactions among types. A collaboration involves two kinds of model constructs: A description of the static structure of the affected objects (including relationships, attributes & operations) (this is called the context supplied by the collaboration) and a description of the sequences of messages exchanged among the objects to perform work (this is called interactions supported by the collaboration)

Parameterized collaboration represents a design construct that can be used repeatedly in different designs. The participants in the collaboration, including the classes, relationships, attributes and operations can be parameters of the generic collaboration. The parameters are bound to particular model elements in each instantiation of generic collaboration. Such a parameterized collaboration is called a DESIGN PATTERN.

Design Pattern - a collaboration (as a complete entity representing a design pattern) is depicted as a dotted ellipse containing the name of the pattern. A dotted arrow is drawn from the collaboration symbol to each of the objects or classes that participate in the collaboration. Each arrow is labelled by a role of the participant.

Context - a context is a view of one or more modelling elements that are related for a particular purpose (such as performing an operation). A context may be a projection from a more complete model, from which details irrelevant to the particular purpose have been surpressed. A context is not a modelling element - but is a term for a fragment of the static model that underlies a collaboration.

Interaction - a collaboration of objects interacts to accomplish a purpose by exchanging messages. The messages may include both signals and calls as well as more implicit interaction through conditions and time events. A specific pattern of message exchange to accomplish a specific purpose is called an interaction.

Collaboration Diagram - A collaboration diagram is a context i.e. a graph of objects and links with message flows attached to it's links. The context of the diagram shows the objects relevant to the performance of an operation, including objects indirectly affected or accessed during the operation.

Implementation diagrams show aspects of implementation, including source code structure and run time implementation structure. They occur in two forms: *component diagrams show the structure of the code itself* and *deployment diagrams which show the structure of the run time system*

Component Diagrams - a component diagram shows the dependencies among software components, including source code components, binary code components and executable components. A s/w module may be represented as a component type.

Component diagram is a graph of components connected by dependency relationships. Components may also be connected to components by physical containment representing composition relationships.

Deployment diagrams - show the configuration of runtime processing elements and software components, processes and objects that live on them. S/w component instances represent run time manifestations of code units.A deployment diagram is a graph of nodes connected by communication associations. Nodes may contain component instances; this indicates that the component lives or runs on the node. Components many contain objects. This indicates that the object is part of the component.

.

Components - A component type represents a piece of software code (source, binary or executable) and may be used to show compiler and runtime dependencies. Components are connected to other components by dashed arrow dependencies A component instances represents a run time code unit and may be used to show code units that have identity at run time (including their location on nodes). A component is shown as a rectangle with one small ellipse and two small rectangles protruding from its side

Nodes - a node is a run time physical object that represents a computational resource, generally having at least a memory and often processing capability as well. A node is represented as a figure that looks like a 3 dimensional cube.

Location of Components and objects within objects - Instances may be located within other instances. For example objects may live in processes that live in components that live on nodes. The location of an instance (including objects, component instances and node instances) within another instances may be shown by physical nesting. Alternatively, an instance may have a property tag 'location' whose value is the name of the containing instance.

References

[Adams-96] E Adams, K Willetts, 'The Lean Communications Provider', McGraw Hill, 1996 ISBN 007 070306x

[Berq-96] *Succeeding in Managing Information Highways*, PRISM Consortium, Springer Verlang, Berquist, A. 1996

[Copl-97] J. Coplien, D. Schmidt, Pattern Languages of Program Design, Addison Wesley, ISBN 0 201 60734 4, 1997

[Corba-95] *Object Request Broker 2.0*. Object Management Group, 1995.

[D4B-98] Prospect Deliverable D4B

[Fowl-97] M Fowler, K Scott, UML Distilled, Applying the standard object modelling language, published by Addison Wesley.

[Gamm-95] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison Wesley, 1995.

[ITU-96] Text of draft recommendation G851-01, Study Group 15, ITU June 1996

[Jacob-92] *Object-oriented software engineering: a use case driven approach,* Jacobson Ivar ACM Press Wokingham Addison-Wesley 1992

[Lewi-95] D. Lewis, T. Tiropanis, L.H. Bjerring, J. Hall, Experiences in Multi-domain Management Service Development, Proceedings of the 3rd International Conference of Intelligence in Broadband Services and Networks, Heraklion, Greece, October 1995, ISBN 3-540-60479-0.

[Mowb-97] T. Mowbray, R. Malveau, CORBA Design Patterns, ISBN 0-471-15882-8, Wiley, 1997

[NMF-92] The "Ensemble" Concepts and Format 025, Issue 1.0, Network Management Forum, Morristown, 1992.

[ODP-94] *Reference Model for Open Distributed Processing*, Part 1 Overview and Part 2 Foundations. ISO/IEC 10746-1 (DIS) & 10746-2 (IS) ITU-T X901 & X902. 1994

[ORDIT-93] *ORDIT* process manual, version 0.5 December 1993

[Hall-96] *'Modelling and Implementing TMN based multi domain management,* PREPARE Consortium, J Hall (ed) 1996 ISBN 3-540-61578-4

[Rumb-91] *Object Oriented Modelling and Design*, J Rumbaugh et al, Prentice Hall 1991.

[TINA-94] *TINA-C Service Architecture*, TINA Baseline document TB_MDC.018_1.0_94, Berndt H, Minerva R,

[TINA-93] *Computational Modelling Concepts*, TINA Baseline document TB_A2.NAT.002_3.0_93, December 1993

[TINA-95] *Information Modelling Concepts*, TINA Baseline document TB_EAC.001_1.2_94, H. Christensen, E. Colban, Version 2.0, April 1995

[TMN-96] Telecommunication Managemement Network, M3000, ITU

[UML-97] Unified Modelling Language, Rational Corp.

[Wade-97] V. Wade, D. Lewis, W. Donnelly, D. Ranc, N. Karatzas, M. Wittig, S. Rao, A design process for the development of muli domain service management systems, published in ACTS Guidelines for ATM deployment and interoperability, Baltzer, June 1997

[Wade-97a] V. Wade, D. Lewis, M. Sheppard, M. Tschichholz, J. Hall, A Methodology for Developing Integrated Multi-domain Service Management Systems**,** publsihed in proceedings of IS&N'97, Springer Verlag

[X722-92] *Information Technology, Sturcture of Management Information, Part 4: Guidelines for the Definition of Managed Objects*, Open Systems Interconnection, 1992