ELSEVIER

Computer Communications 24 (2001) 394–415

computer communications

www.elsevier.com/locate/comcom

An integrated and systematic approach for the development of telematic services in heterogeneous distributed platforms

D.X. Adamopoulos[a,*], G. Pavlou[a], C.A. Papandreou[b]

[a]*Centre for Communication Systems Research (CCSR), School of Electrical Engineering, IT and Mathematics, University of Surrey, Surrey, UK*
[b]*Hellenic Telecommunications Organisation (OTE), Athens, Greece*

Received 18 May 1999; revised 22 May 2000; accepted 23 May 2000

Abstract

The advent of deregulation combined with new opportunities opened by advances in telecommunications technologies has significantly changed the paradigm of telecommunications services, leading to a dramatic increase in the number and type of services that telecommunication companies can offer. Building new advanced multimedia telecommunications services in a distributed and heterogeneous environment is very difficult, unless there is a methodology to support the entire service development process in a structured and systematic manner, and assist and constrain service designers and developers by setting out goals and providing specific means to achieve these goals. Therefore, in this paper, after a brief examination of important service engineering matters and service modelling issues, a service creation methodology a proposed and presented focusing on its essential characteristics. The application of this methodology to the development of a multimedia conferencing service for education and training is then examined and a possible enhancement of the methodology through the use of design patterns and frameworks is considered finally.

Keywords: Service development methodology; Service creation; Service engineering; New telecommunications services; Service life-cycle

1. Introduction

The telecommunications industry is currently undergoing a fundamental restructuring as the evolving synergy between information and telecommunications technology, termed telematics, is gradually gaining momentum. This evolution and diversification is being driven mainly by liberalisation, increasing competition in the marketplace, technological advancements, and demands from all customer segments for an increasingly sophisticated portfolio of telecommunication services, tailored to their specific needs. It is expected that the forthcoming integrated (fixed and mobile) broadband networks will be available openly to the existing and the new service providers, consisting of a world-wide common shared communications platform for a multitude of new advanced telecommunications services (telematic services). The proliferation of these services will lead eventually to the transformation of the global information infrastructure to an open services market fuelled by deregulation, strategic partnerships, and joint interoperability activities [23].

There is much incentive to stay ahead of this global market, and offer new and/or improved services before the competition. Pressure on service providers is increased as they need to be able to react quickly and flexibly to the ever changing customer needs by developing and offering services of enhanced functionality and significant diversity in shorter time-frames. Therefore, the rapid deployment of new or improved services is critical, and the service life-cycle has to accelerate so that new services can be provided fast enough to meet the changing customer demands in a competitive manner. However, the fast and cost-effective provision of the new efficient services requires not only an open service architectural framework, like the one specified by the Telecommunications Information Networking Architecture Consortium (TINA-C) [5,27], but also appropriate support for the service development process [2,11].

The creation of telecommunications services within an open environment is a highly complex activity. This complexity stems not only from the technical nature of the tasks involved, but also from the number of the participating actors and the variety in their roles, concerns, and skills. Therefore, there is a need to support the complex service

* Corresponding author.
E-mail addresses: d.adamopoulos@eim.surrey.ac.uk (D.X. Adamopoulos), g.pavlou@eim.surrey.ac.uk (G. Pavlou), kospap@org.ote.gr (C.A. Papandreou).

0140-3664/01/$ - see front matter
PII: S0140-3664(00)00230-9

creation process in order to ensure that resulting services actually perform as planned and as required by customers and service providers [12]. A methodology is an important part of such an attempt, as it provides a systematic and structured base for the flexible and efficient management of the development of telecommunications services, also ensuring that the roles of the participating actors are identified clearly and that their behaviours are consistent throughout the whole process of service creation.

In this paper, in order to structure and control the service development process from requirements capture and analysis to service implementation, to reduce the inherent complexity, and to ensure the thorough compatibility among the many involved tasks, a TINA-C conformant service creation methodology is proposed. Its intention is to provide valuable answers to several important service engineering matters and thus facilitate the transition to a telecommunications environment where many different (enhanced) services are offered by a multiplicity of service providers to several categories of customers within an open market.

2. The need for an integrated and systematic approach

An architectural framework is by its definition an abstract entity, which consists of a set of concepts/principles and a set of guidelines and rules. For this reason, TINA-C is more descriptive rather than prescriptive, and its application can be a complex task [26]. Furthermore, there seems to be no end to the emergence of new services, each requiring a new set of communications capabilities. In a world already replete with a multitude of services, the addition of new intricate services can be a daunting challenge.

The basic factors that shape this challenge are addressed by the discipline of integrated service engineering, which includes the technologies and engineering processes required to define, design, implement, test, deploy, maintain, and manage telecommunications services [2,3]. The attribute "integrated" makes reference to the need for information, management, service, component, and interface integration (integration aspects), required to support effectively new telecommunications services.

The concept of the service life-cycle has a central role in integrated service engineering. All services go through a service life-cycle, which contains descriptions of activities, in the form of an ordered collection of processes or steps, that are required to support the development, the operation, and the maintenance of a service [9]. The logical grouping of these activities gives rise to a number of distinct sets, which are known as stages (service creation, service deployment, service operation/utilisation, and service withdrawal). Further grouping of the activities within a stage gives rise to the concept of actions (or phases). The description of the actions includes all the essential details of the activities that take place in a given stage. The service life-cycle establishes a common terminology to be used when discussing a service, and thus facilitates a common understanding of service matters.

Fig. 1 depicts a graphical representation of a service life-cycle, which is an enriched variation of the TINA-C life-cycle model [27]. The rectangles are the actions/phases, while the ellipses are the main states that a service goes through. In this figure, the following states are identified:

- *Conceived but not planned*—the service has been conceived, but no details about its implementation are known.
- *Planned but not installed*—the service has been planned, but it does not exist in a (TINA-C) service execution environment (although it might have existed in the past).
- *Installed but not activated*—the components of the service exist in a (TINA-C) service execution environment, but the service cannot be instantiated.
- *Activated but not instantiated*—the service has the potential for being instantiated.
- *Instantiated*(*executing*)—an instance of the service is available.

The service life-cycle of Fig. 1 is a combination of traditional software engineering methodologies (focusing on development issues) with the activities required to operate, use, and maintain a service (focusing on post-development issues) [10]. It should be noted that it is not a strict waterfall model (it is not a strict top–down approach) of system development. It is possible in each phase of the life-cycle (and especially in the service creation phases) to return to a previous phase if the refinements and requirements are added during system development.

Among all the stages of the service life-cycle of Fig. 1, in TINA-C, service creation is one of the most abstract and general, since it does not provide many guidelines on how to structure each of its phases. Furthermore, it is also one of the most important as it determines the efficiency with, which the services will be developed, and thus the success of service providers in a highly competitive market. For this reason, considerable effort is being devoted in Europe to the definition of advanced service creation practices and the development of Service Creation Environments (SCEs) [1,30].

In order to meet these challenges, a service creation methodology is proposed to enable TINA-C to satisfy the required expectations on long-term efficiency of service design, provision, and management. This methodology, given a set of requirements that a service should meet, a set of the available service independent features (normally in the form of service components), and a target TINA-C compliant Distributed Processing Environment (DPE) wherein the service will be deployed, facilitates the design and implementation of a TINA-C compliant service, which meets the desired requirements by promoting the use of the service independent features [14,25].

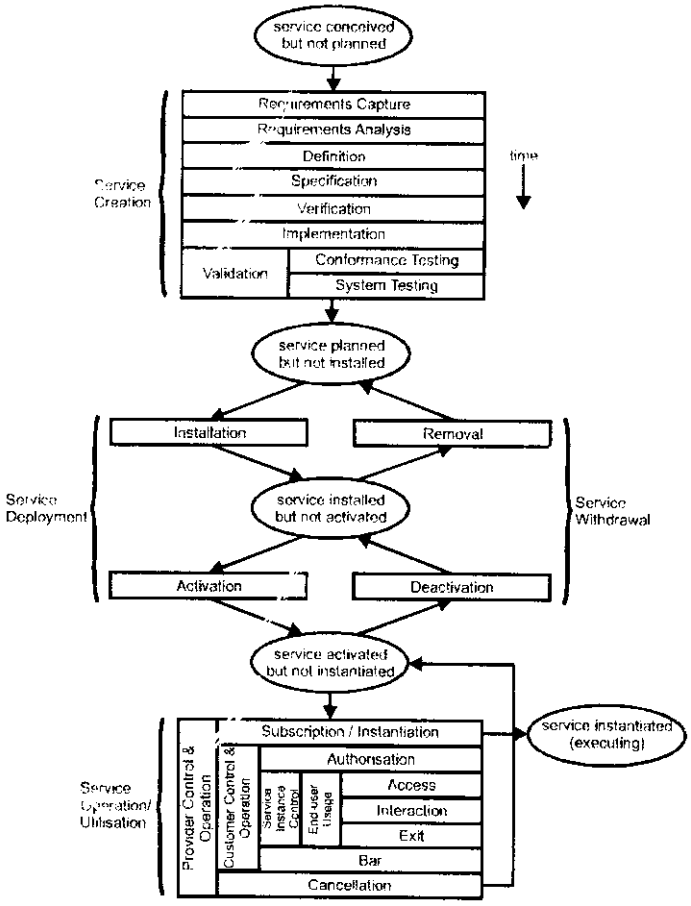

Fig. 1. The service life-cycle.

Such a methodology can result in an efficient and effective standardised process for the systematic development of telematic services, at lower cost, higher quality, increased speed, meeting the needs of the users and the enterprise better, and leading to lower maintenance costs. Moreover, a common approach can be used throughout an enterprise, which means that more integrated systems can result, the management of related projects will be facilitated significantly, staff can change easily from project to project without much retraining, the reuse of service specifications and software will be supported greatly, and a base of common experience and knowledge will be achieved [2,10].

3. The modelling approach

A telematic service, due to its (potentially) enhanced functionality and its inherent distributed nature, is usually overwhelmingly complex and thus, it is necessary to decompose it into understandable parts/segments in order to comprehend fully its semantics and manage the complexity. These parts/segments may be represented as models that describe and abstract essential aspects of the telematic service.

Therefore, a useful activity during the development of a telematic service is to create models of the service, which organise in a concise way and communicate with accuracy the important details of the telematic service under examination. These service models should contain cohesive, strongly related elements and are composed usually of other (simpler) models or artifacts, comprising basically diagrams and documents, which describe concepts and entities, and reveal the relations between them.

Service models in the proposed methodology are presented using the Unified Modelling Language (UML), which is an emerging industry standard for specifying, constructing, visualising, and documenting software-intensive systems [6,15]. UML is an elegant, expressive, and flexible object-oriented modelling language, capable to support effective and consistent communication and to enhance the ability to understand and act. Although UML must be applied in the context of a process, it does not define a standard development process, as it has been experienced that different organisations and problem domains require different processes. Therefore, UML consists only of a metamodel (which unifies semantics) and a notation (which provides a human rendering of these semantics).

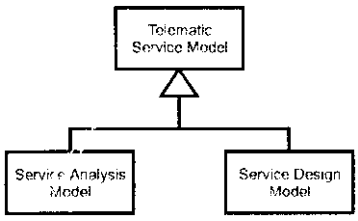

Fig. 2. The overall telematic service model.

UML was chosen as the main modelling notation for the proposed methodology because:

- It provides service designers and service developers a ready-to-use, expressive, visual modelling language so that they can develop and exchange meaningful service models.
- It provides extensibility and specialisation mechanisms to extend the core concepts and thus it can be tailored to the specific needs of telematic services.
- It is independent of particular programming languages and development processes.
- It provides a formal basis for understanding the modelling language.
- It encourages the growth of the object-oriented software tools market and thus the availability of appropriate tool support for the entire service creation life-cycle is guaranteed.
- It supports useful higher-level development concepts, such as collaborations, frameworks, (design) patterns, and components.
- It integrates best practices from popular first-generation object-oriented analysis and design methods.

The overall telematic service model that is created when applying the proposed methodology to the development of a specific telematic service can be seen in Fig. 2 and is composed of the:

- *Service Analysis Model*—it is related to an investigation of the domain of the telematic service (service domain), which constitutes the problem space under examination.
- *Service Design Model*—it is related to a (logical) specification of the constituent parts of the telematic service.

In the proposed methodology, independent of how artifacts are organised into service models, there are influential dependencies between them. It is useful to understand these dependencies so that consistency checks and traceability can be achieved, and so that dependent artifacts are used effectively as input to creating later artifacts. If an artifact is created, which has no dependants and is not used as input to anything else, then the value of that artifact and the time spent in its creation should be questioned seriously [21].

Finally, it has to be stressed that the overall modelling approach followed by the proposed methodology is influenced strongly by the TINA-C service architecture and by the modelling guidelines and rules that it encompasses [22,27]. Services are considered as software-based applications that operate on a distributed computing platform (a DPE). This platform hides from services the underlying technologies and distribution concerns, and supports in this way the portability and interoperability of the service code. Therefore, a telematic service is realised by a set of interacting service components (i.e. computational objects interacting via their computational interfaces), which are distributed across different network elements.

4. The proposed service development methodology

Telecommunications operators need to master the complexity of service software, because of the highly diversified market demands, and consequently, because of the necessity to develop quickly and economically and introduce a broad range of new services [10]. To achieve such an ambitious, yet strategic to the telecommunications operators goal, a service creation methodology based on the rich conceptual model of TINA-C is proposed.

A methodology is considered to be a coherent and integrated set of methods/procedures from, which a coherent subset can be selected by developers for particular cases. A method/procedure is a systematic way to achieve a specific goal. It is implemented by techniques, and some techniques are supported, or automated, by tools [2,9]. In full agreement, the proposed methodology contains a conceptual model of constructs and a series of guidelines, essential to the development of telematic services, together with a set of (partially) ordered activities (realised by a number of specific steps) suggesting the direction to proceed.

A high-level or macro-level view of the proposed service creation methodology can be seen in Fig. 3. The proposed service development process is based on an iterative and incremental, use case driven approach. An iterative service creation life-cycle is adopted, which is based on successive enlargement and refinement of a telematic service through multiple service development cycles within each one the telematic service grows as it is enriched with new functions. More specifically, after the requirements capture and analysis phase, service development proceeds in a service formation phase, through a series of service development cycles. Each cycle tackles a relatively small set of service requirements, proceeding through service analysis, service design, service implementation, and service validation and testing. The telematic service grows incrementally as each cycle is completed. The proposed use of the UML notation in almost all the phases of the methodology, has the advantage of making both the service description more coherent and the process of proceeding from one phase to another more natural and efficient [18,21]. For this reason, TINA-C

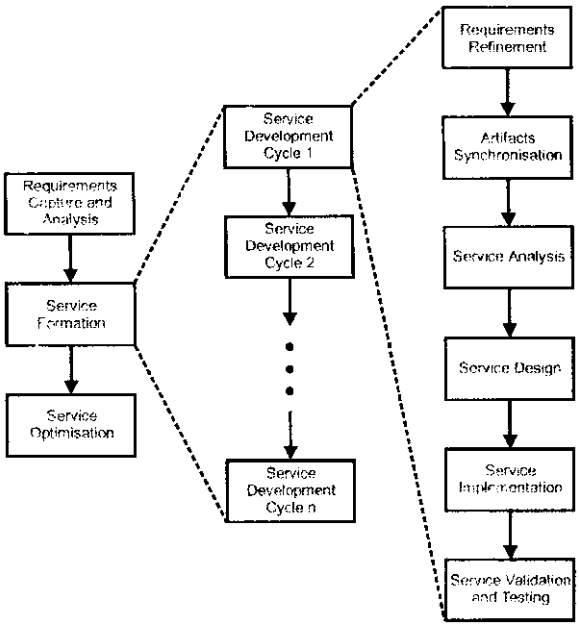

Fig. 3. Iterative service development cycles in the proposed methodology.

could also consider UML in a future version of its computing and service architectures.

According to Fig. 3, the main phases of the proposed methodology are the following:

- *Requirements capture and analysis phase*—it identifies the telematic service requirements (together with a number of roles) and presents them in a structured way.
- *Service analysis phase*—it describes the semantics of the problem domain that the telematic service is designed for. Thus, it identifies the objects that compose a service (information service objects), their types, and their relationships.
- *Service design phase*—it produces the design specifications of the telematic service under examination. Computational modelling is taking place in this phase and thus the service is described in terms of (TINA-C) computational objects interacting with each other.
- *Service implementation phase*—in this phase the pieces of the service software (computational objects) are defined and implemented in an object-oriented programming language (e.g. C++, Java), inside a TINA-C compliant DPE.
- *Service validation and testing phase*—it subjects the implemented telematic service in a variety of tests in order to ensure its correct and reliable operation.
- *Service optimisation phase*—it examines thoroughly the service code in order to improve its performance in the target DPE and thus prepares the telematic service for a successful deployment.

As can be seen from Fig. 3, the proposed methodology is consistent conceptually with the viewpoint separation as advocated by TINA-C in accordance with the Reference Model for Open Distributed Processing (RM-ODP), uses the service life-cycle of Fig. 1 as a roadmap, and does not imply a waterfall model in which each activity is done once for the entire set of service requirements. Furthermore, graphical and textual notations are proposed for almost all phases to improve the readability of the related results and ensure a level of formalism sufficient to prevent any ambiguity. In the following paragraphs, a number of important issues regarding the proposed methodology as a whole are considered and the most important phases of the methodology are examined focusing on their essential characteristics and artifacts. The service optimisation phase has been omitted, because it depends highly on the selected programming language and on the target DPE. Finally, an attempt to apply all the phases of the methodology to the development of a specific characteristic telematic service is presented and critically analysed.

4.1. Important methodological considerations

The proposed methodology specifies a service development process from the identification of service requirements through to the actual implementation of a telematic service. This service development process, except from the fact that it organises all the activities related to the creation of telematic services, it also provides a foundation for creating a manageable, repeatable, and successful service development project.

The most important phases of the proposed methodology are the requirements capture and analysis phase, and the phases of service analysis and service design, which are performed inside the repeated service development cycles. A useful approach is to bind each of these cycles within a rigidly fixed time-frame (a time-box). All work related to a specific cycle must be accomplished in that time-frame. Depending on the telematic service under examination, a time-frame ranging between two weeks and two months is usually appropriate. Any less, and it is difficult to complete tasks; any more and the complexity becomes overwhelming and feedback is delayed [8,21].

To succeed with a time-box schedule, it is necessary to choose the service requirements carefully, as iterative service development cycles are organised according to the identified use case requirements. More specifically, a service development cycle is assigned to implement one or more (semantically related) use cases, or simplified versions of use cases (which is quite common when the complete use case is too complex to tackle in one cycle). Use cases should be ranked and high ranking use cases should be tackled in the early service development cycles. Furthermore, use cases that (is expected to) significantly influence the core service architecture, or which are critical and/or high-risk, should be considered first.

To create a telematic service successfully, a clear description of the problem and of the service requirements

is necessary (i.e. what the problem is about and what the telematic service must do). It is also necessary to have high-level and detailed descriptions of a logical solution, which fulfils the identified service requirements and satisfies any potential constraints. Service analysis emphasises an investigation of the problem rather than how a solution is defined, while service design emphasises the construction of a logical solution according to the service requirements. Ultimately, the service specifications produced by the service design phase will be implemented in the software with the appropriate use of computing and networking infrastructure.

Service development is complex and therefore decomposition ("divide-and-conquer") is the primary strategy considered to deal with this complexity by breaking a telematic service up into manageable units. The proposed methodology applies object-oriented analysis and design, which emphasises considering a problem domain and an associated logical solution from the perspective of objects (things, concepts, or entities) [3,6]. This is opposed to decomposing a telematic service by applying structured analysis and design, whose dimension of decomposition is primarily by function or process, resulting in a hierarchical breakdown of processes composed of subprocesses.

Because of the adoption of object orientation by the proposed methodology, during the (object-oriented) service analysis phase, there is an emphasis on finding and describing the service concepts (service information objects) in the service domain, whose boundaries are determined, as accurately as possible, by the telematic service under examination during the requirements capture and analysis phase. Furthermore, during the (object-oriented) service design phase, there is an emphasis on defining logical (software) service objects (service computational objects or service components) with attributes and methods that will ultimately be implemented in an object-oriented programming language, such as C++ or Java, inside a TINA-C compliant DPE during the service implementation phase.

It has to be stressed that the division between service analysis and service design is fuzzy. In telecommunications service engineering (and sometimes in other disciplines also), analysis and design work exists on a continuum, and different practitioners of service analysis and service design classify an activity at varying points on the continuum [10,25]. Therefore, it is not helpful being rigid about what constitutes a service analysis versus a service design step.

Nevertheless, some consistent distinction is useful in practice between investigation (service analysis) and solution (service design), because it is advantageous to have a well-defined step that emphasises an inquiry of what the problem is before examining in detail how to create a solution. Finally, if a team of service developers is involved in the service development process, a consistent distinction between service analysis and service design sets an expectation of suitable behaviour among the team members. For example, during service analysis, members expect to emphasise understanding of the problem while deferring issues related to the actual solution, performance, etc.

Irrespective of the exact scope of service analysis and service design, the most important ability in both these two phases is to assign responsibilities to service components successfully [9,21,30]. This is the most critical skill, because this activity must be performed (it is inescapable) and it has the most profound effect on the robustness, maintainability, and reusability of the resulting (implemented in software) service components, which are the main building blocks of telematic services. Assigning responsibilities is inevitable even when a service developer hasn't got the opportunity to perform any other service analysis or service design activities (a "rush to code" service development process). Next to assigning responsibilities, another important activity is finding suitable service objects or abstractions. Both activities are critical, but responsibility assignment tends to be the more challenging skill to master, as it is difficult to devise and apply guidelines for this activity.

Finally, the timing of the creation of some artifacts during the application of the proposed methodology needs to be discussed and examined. More specifically, certain artifacts created during the service analysis phase, such as a service conceptual model and expanded use cases, also may be created during the requirements capture and analysis phase.

A service conceptual model is a graphical representation of service concepts in the service domain under examination (see also Section 4.3). The amount of effort applied to the creation of a draft service conceptual model during the requirements capture and analysis phase needs to be tempered. The goal is to obtain a basic understanding of the vocabulary and concepts used in the service requirements. Therefore, a fine-grained investigation is not required, as it also increases the possibility of front-loading the investigation too much (complexity overload).

The recommended strategy is to create quickly a rough service conceptual model where the emphasis is on finding obvious service concepts expressed in the service requirements while deferring a thorough investigation. Later, within each service development cycle, the service conceptual model will be refined incrementally and extended for the service requirements under consideration within that cycle. Another possible strategy is to defer creation of the service conceptual model completely until the start of the service development cycles. This has the advantage of deferring complexity, but the disadvantage of less up-front information, which may have been useful for achieving better comprehension.

Another artifact that the timing of its creation needs to be examined in use cases (see also Section 4.2). During the requirements capture and analysis phase, it is proposed to create all the high-level use cases, but to only rewrite the most critical and important use cases in an expanded (long) format, deferring the rest until the service development cycle in which they are examined. As with the service

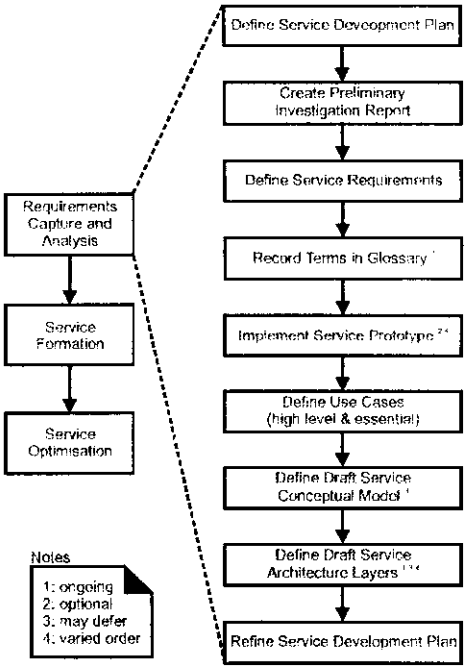

Fig. 4. Requirements capture and analysis phase activities.

conceptual model, there are trade-offs in terms of the benefit of the early acquisition of information versus facing too much complexity. The advantage of investigating and writing all the detailed expanded use cases during the requirements capture and analysis phase is more information, which can improve comprehension. However, the disadvantage is early complexity overload, as this investigation will generate many more detailed issues. Furthermore, the expanded use cases may not be very reliable because of incomplete or incorrect information, and because the service requirements may be under continual change. Therefore, the recommended strategy is to expend effort investigating only the most important use cases in detail during the requirements capture and analysis phase.

4.2. Requirements capture and analysis

During this phase the service developer assembles, documents, and structures the requirements on the service (service needs) from the different stakeholders involved. The focus is on modelling the concepts that are visible at the service boundary and thus the service logic is viewed as a black box. The requirements capture and analysis phase is a critical phase because the correct and thorough specification of the service requirements is essential for a successful telematic service. The primary goal of this phase is to identify what functionality is really needed to include in the telematic service and document it in a form that is easily understandable and unambiguous.

The activities that take place in this phase can be seen in Fig. 4. Although this figure suggests a linear order of artifact creation, that is not strictly the case. Some artifacts may be

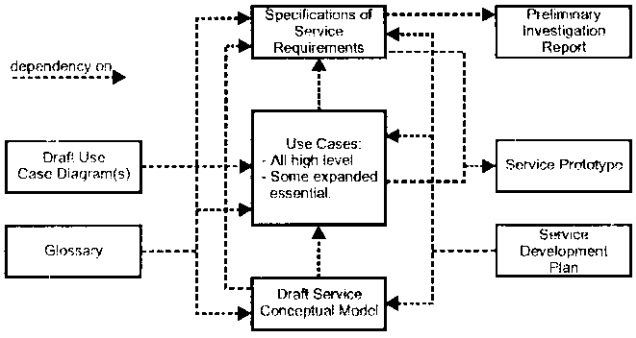

Fig. 5. Requirements capture and analysis phase artifact dependencies.

created in parallel. This is especially true for the draft conceptual model, the glossary, the use cases, and the use case diagram(s). The dependencies between the artifacts produced during the requirements capture and analysis phase can be seen in Fig. 5.

The service developer, after gathering enough material regarding the telematic service under examination with various means (e.g. interviews, group meetings, study of related documents, etc.) and in various forms/formats (e.g. notes, sketches/graphs, audio recordings, etc.), attempts to process this material and structure it appropriately in order to elicit from it a set of service requirements.

In full agreement, one of the most important tasks that the service developer has to perform is the identification of the independent entities/actors, which are involved (by their collaboration) in the operation of the service within and across business administrative domain boundaries. These entities correspond to roles modelling a well defined grouping of functionality under control of a specific stakeholder [23]. This initial task is important because the gathered requirements are structured around the identified roles by determining relationships between the roles. Each of the relationships define a set of specifications for the interactions between two roles. They also support generic interactions (e.g. instance establishment, release and management of a secure association, negotiation of the initial usage interactions, etc.) that need to be performed before any other interaction defined in the relationship can occur. A role can be either generic or specific. The main generic (business) roles and the (business) relationships between them are specified by the TINA-C Business Model [28], which can be seen in Fig. 6. Each generic role (consumer, retailer, broker, third party service provider, connectivity provider) corresponds to one or more specific roles [22].

TINA-C reference points are defined in relation to the (business) relationships they support and, according to their functionality, can be divided into the following segments, which cover the core parts of the functionality of a TINA-C service [5,27]:

- *Access segment*—it is concerned with authentication and authorisation of users, the selection of service features,

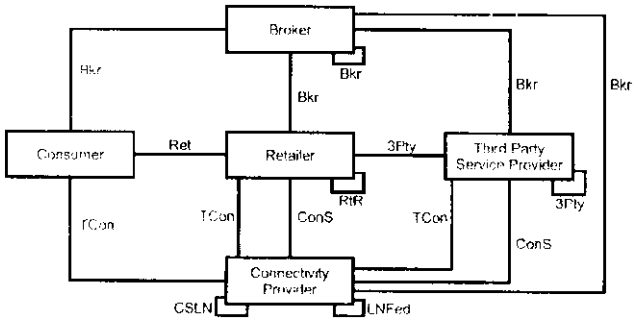

Fig. 6. The TINA-C Business Model (roles and relationships).

and the setting up of the context for the use and management of the service.

- *Usage segment:*
  - *primary*—it covers the functionality that is the main objective of the service;
  - *ancillary*—it addresses administrative and management functionality.

In order to improve the understanding of the service requirements, use cases are created. Use cases are textual narrative descriptions of service domain processes. They describe the sequence of events generated by an actor using a telematic service to complete a specific service process. They are stories or cases of using a telematic service. Use cases are not exactly requirements or functional specifications by themselves, but they illustrate and imply service requirements in the stories they present [8,15]. Use cases may be expressed with varying degrees of detail and commitment to design decisions. Therefore, the same use case may be written in different formats, with different levels of detail. There are two basic formats that a use case can take, leading, respectively, to high-level use cases and to expanded use cases.

A high-level use case describes a service process very briefly, usually in two or three sentences. It is useful to create this type of use case during the requirements capture and analysis phase in order to understand quickly the degree of complexity and functionality in a telematic service. High-level use cases are very terse and vague on design decisions. On the other hand, an expanded use case describes a service process in more detail than a high-level one. The primary difference from a high-level use case is that it has a section, which describes the step-by-step events. During the requirements capture and analysis phase, it is useful to write the most important and influential use cases in the expanded format, and defer the less important ones until the service development cycle in which they are being tackled (see also Section 4.1).

Furthermore, use cases can be either essential or real. More specifically, essential use cases are expanded use cases that are expressed in a form that remains relatively free of technology and implementation details, as design decisions (especially those related to the user interface) are deferred and abstracted. High-level use cases are always essential in nature, due to their brevity and abstraction. It is desirable to create essential use cases during the requirements capture and analysis phase in order to more fully understand the scope of the problem and the service functions required. They are advantageous because they reveal the essence and the fundamental motivation of the service process that they describe without being overwhelmed with design details. They also tend to be correct for a long period of time, since they exclude design decisions [8].

In contrast, a real use case concretely describes a service process in terms of its real current design and committed to specific technologies. Ideally, real use cases are created during the service design phase of a service development cycle, since they are a design artifact. If early design decisions regarding the user interface are expected, then real use cases must be created during the requirements capture and analysis phase. Otherwise, it is undesirable to create real use

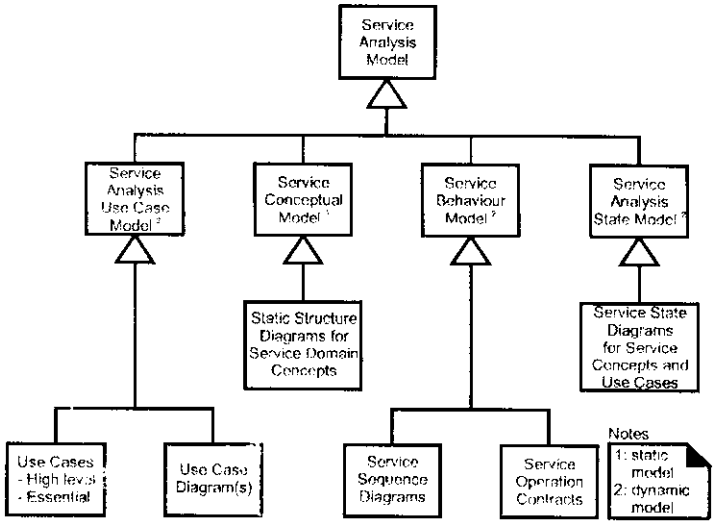

Fig. 7. The service analysis model.

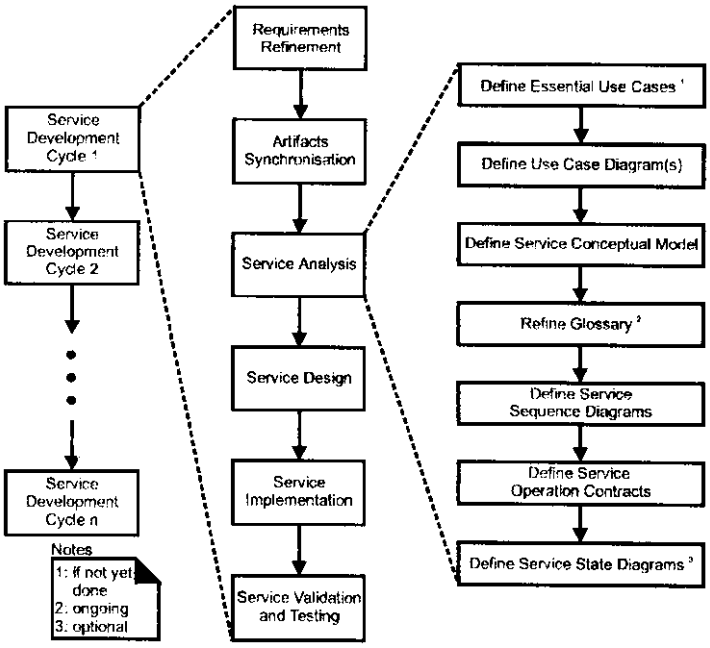

Fig. 8. Service analysis phase activities.

cases so early, because of the premature commitment to a specific service design and overwhelming complexity involved.

After the identification of use cases and concurrently with their specification, a use case diagram is created. It illustrates a set of use cases for a telematic service, the actors involved, and the relation between the actors and the use cases. The purpose of this diagram is to present a kind of context diagram by which one can understand quickly the external actors of a telematic service and the key ways in which they use it. High-level and essential use cases and use case diagrams are members of the service analysis use case model (see Fig. 7).

4.3. Service analysis

The aim of this phase is to determine the functionality needed for satisfying the service requirements that were identified in the previous phase and to define the software architecture of the service implementation. For this reason, the focal point shifts from the service boundary to the internal service structure [2].

The activities that take place in this phase can be seen in Fig. 8. As with the requirements capture and analysis phase artifacts, the linear order that may be inferred from this figure is not strictly the case, as some artifacts may be created in parallel (e.g. the service conceptual model and the glossary). The dependencies between the artifacts produced during the service analysis phase can be seen in Fig. 9.

The service analysis phase is the first phase of the service creation process where the service is decomposed into constituent parts (service information objects or service concepts), with the appropriate relationships among them, in an attempt to gain an overall understanding of the service. The resulting (main) service conceptual model, which is the most important artifact that is created during the service analysis phase, represents a restatement, in a graphical notation, of the problem statement, as it was expressed in the previous phase. It involves identifying a rich set of service concepts regarding the telematic service under examination by investigating the service domain. Therefore, it describes what the service is in terms of interesting and meaningful (to the service developer) entities/concepts that constitute it and the couplings/associations between them. These couplings define relationships between two service Information Object (IO) classes. Each service IO participating in a relationship has a role in that relationship. Each role possesses a certain multiplicity that quantifies the number of instances of a service IO class having a role that may participate in a relationship with each instance of the service IO class having the other role [16]. In UML, a service conceptual model can be illustrated with a set of static structure diagrams in which no operations are defined.

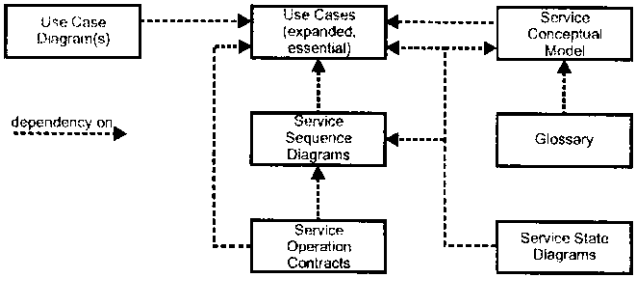

Fig. 9. Service analysis phase artifact dependencies.

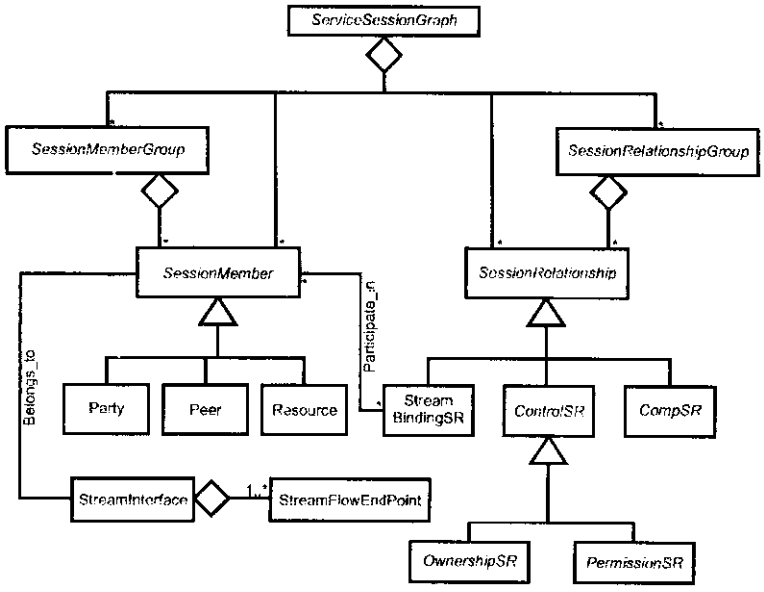

Fig. 10. The Service session graph.

It has to be stressed that a service conceptual model is a representation of real-world concepts or actual things, and not a representation of software components (software entities). A good service conceptual model captures the essential abstractions and information required to understand the service domain in the context of the current service requirements, and aids service developers in understanding the service domain (its concepts, terminology, and relationships). However, there is no such thing as a single correct service conceptual model. All service conceptual models are approximations of the service domain under examination [6,21].

A central task when creating a service conceptual model is the identification of the service concepts (mainly by examining expanded use cases). It has to be noted as a guideline that it is better to overspecify a service conceptual model with many fine-grained concepts than to underspecify it. It is also necessary to identify those associations of service concepts that are needed to satisfy the information requirements of the current use case(s) under development and that aid the comprehension of the service conceptual model. The associations that should be considered, in order to be included in a service conceptual model, are those associations for which the service requirements suggest or imply that knowledge of the relationship needs to be preserved for some duration ("need-to-know" associations) or are otherwise suggested strongly in the service developer's perception of the problem domain. Finally, in a similar manner, a service conceptual model should include all the attributes of service concepts for which the service requirements suggest or imply a need to remember information.

The main service conceptual model is accompanied by a set of ancillary service conceptual models. These models are derived by (and correspond to) a number of generic information models deduced from the TINA-C service architecture and complement semantically the main service conceptual model with useful session related concepts and structures. More specifically, the ancillary models refer to the modelling of session roles, (TINA-C) sessions, access sessions and service sessions, and to the classification of access and service sessions. The most important of them is the Service Session Graph (SSG), which offers a generic framework to describe information in service sessions and is used to model and control the state of a service session [27]. An instance, at a certain point of time, of the SSG models a "snapshot" of the resources, the parties, the peers, and the relationships established into the service session. The capabilities modelled in the SSG, which can be seen in Fig. 10, are party invitation and addition, stream binding and stream composition, and explicit control of the use of resources (e.g. various devices participating in continuous media communication).

The SSG supports the definition of control relationships through the `ControlSR` type. More specifically, the `OwnershipSR` class expresses ownership relations, which determine the owners of a service IO that need to be involved in the negotiation of session management operations. Furthermore, the `PermissionSR` class expresses the desired access control policy. `SessionMember` and `SessionRelationship` IOs can be aggregated into groups (`SessionMemberGroup` and `SessionRelationshipGroup`, respectively) in order to ease the establishment of repeated relationships. The invitation and addition of users to a service session is modelled by the Party IO type. Stream bindings among parties are modelled by the association of `StreamInterface` IOs to `StreamBindingSR` IOs via the appropriate `SessionMember` IOs. The `StreamFlowEndPoint`

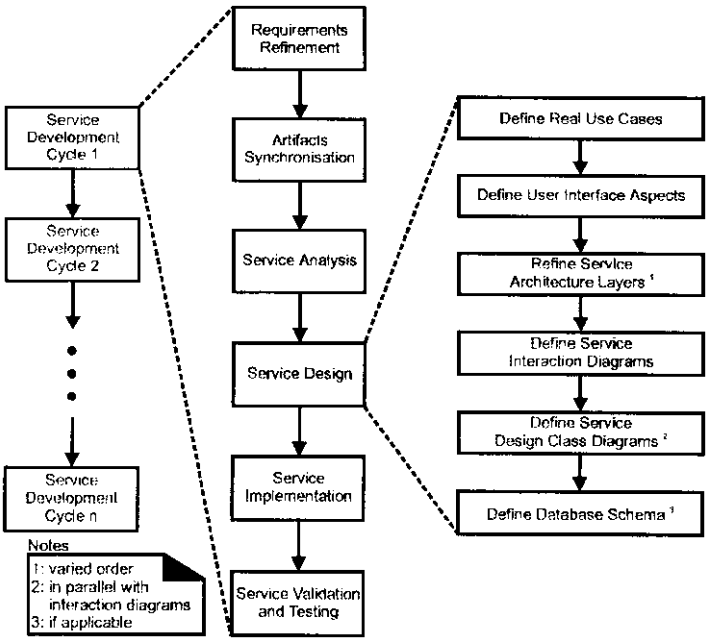

Fig. 11. Service design phase activities.

IOs have associated Quality of Service (QoS) attributes and can be aggregated into stream interfaces at a party's end system or at a `Resource` IO.

Before proceeding to a logical design of how a telematic service will work in terms of software components, its behaviour is necessary to be examined and defined as a black box. In this way, service behaviour is considered as a description of what the telematic service does, without explaining how it does it. One part of that description are service sequence diagrams.

Use cases suggest how actors interact with the telematic service under examination. During this interaction, an actor generates events to the telematic service, requesting some operation in response. It is desirable to isolate and illustrate the operations that an actor requests of a telematic service (service operations), because they are an important part of understanding service behaviour. A service sequence diagram, which is a UML sequence diagram, shows, for a particular scenario of a use case, the events that external actors generate and their order. All telematic services are treated as a black box. The emphasis of the diagram is on events that cross the service boundary from actors to telematic services. A service sequence diagram should be done for the typical course of events of each use case and sometimes for the most important alternative courses.

The behaviour of a telematic service is further defined by service operation contracts (or service contracts), as they describe the effect of service operations upon the telematic service. A service sequence diagram depicts the external events that an actor generates, but it does not elaborate on the details of the functionality associated with the service operations invoked. All the details that are necessary to understand the service response (and thus the actual service behaviour) are missing. These details are included in service operation contracts, which describe changes in the state of the overall telematic service when a service operation is invoked. UML contains support for defining service contracts by allowing the definition of pre- and post-conditions of service operations [15].

During the service analysis phase it is difficult (and maybe it is not even necessary) to generate a complete and accurate set of post-conditions for a service operation. However, it is better to create post-conditions early (even if they are incomplete), rather than defer their creation until the service design phase, when service developers should be concerned with the design of a solution, rather than investigating what should be done. Post-conditions will take their final form during the service design phase. Therefore, they will enhance the service analysis work of the following service development cycle.

Service sequence diagrams and service operation contracts are part of the service behaviour model of the service analysis model (see Fig. 7), which specifies what service events a telematic service responds to, and what responsibilities and post-conditions the corresponding service operations have. The service behaviour model describes the external interface and the behaviour of the overall service. It has to be noted that in order to be complete (and really object-oriented) the information specification of the service should also take into account the (dynamic) behaviour of individual service IOs. This behaviour is defined usually by allocating operations to the service IOs. However, the issue whether operations should be ascribed to individual service IOs is quite controversial as it actually represents a functional decomposition of the overall service functionality [16]. This clearly implies

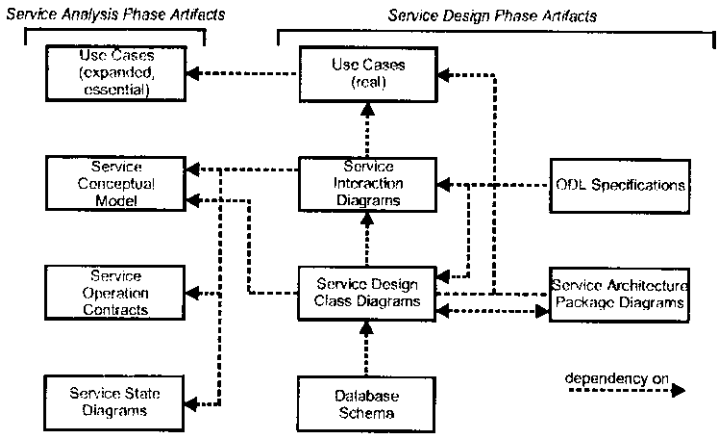

Fig. 12. Service design phase artifact dependencies.

design decisions that should be better taken at the service design phase.

4.4. Service design

During this phase the service developer defines the behaviour of the service IOs that were identified in the service analysis phase and structures the telematic service in terms of interacting service computational objects (service components), which are distributable, multiple interface service objects. They are the units of encapsulation and programming. While service IOs mainly explain how a service is defined, service Computational Objects (COs) reveal what actions have to be performed in order to execute the service. Therefore, the output of this phase is the dynamic view of the internal structure of the telematic service.

The activities of the service design phase are depicted in Fig. 11. As with the previous phases, the linear order that may be inferred from this figure is not strictly the case, as some artifacts may be made in parallel (e.g. the service interaction diagrams and the service design class diagram). The dependencies between the artifacts produced during the service design phase can be seen in Fig. 12. This figure also shows the way that the service design phase artifacts depend on some of the service analysis phase artifacts.

As a first step in this phase, the service IOs are considered as potential candidates for service COs. In many cases, service IOs are mapped to one corresponding service CO encapsulating the information defined by the service IO and providing an operational interface to access that information. However, the mapping between service IOs and service COs is not necessarily one to one. Furthermore, the existence of a relationship between service IOs, either provides a good rationale for encapsulating them together in the same service CO or indicates the need for a binding between interfaces of their corresponding service COs [10,11]. This mapping process is simplified significantly by adopting the use of the generic (access session, service session, and communication session related) COs proposed by the TINA-C service architecture (in terms of their identified functionality and not in terms of specific interfaces-feature sets) and by considering the computational views of a number of scenarios (regarding business administrative domains in user-provider roles and peer-to-peer access roles, and in compound service sessions) deduced by the computational modelling guidelines of TINA-C, which are useful (for improving structure and general comprehension) throughout the service design phase.

After identifying the service COs, a (separate) service interaction diagram is created for each service operation under development in the current service development cycle. Service interaction diagrams illustrate how service objects communicate in order to fulfil the service requirements. More specifically, the expanded use cases suggested the service events initially, which were explicitly shown in service sequence diagrams, then an initial best guess at the effect of these service events was described in service operation contracts, and finally the identified service events represent messages that initiate service interaction diagrams, which illustrate how service objects interact via messages to fulfil the required tasks. The service designer may collect/extract information about what tasks the service interaction diagrams should fulfil by essential or real uses cases, and by the post-conditions of the service operation contracts. However, it is essential to recognise that the previously (in the service analysis phase) defined post-conditions are merely an initial best guess or estimate of what must be achieved and they may not be accurate.

UML defines two kinds of interaction diagrams, either of which can be used to express similar or even identical message interactions; namely collaboration diagrams, which illustrate object interactions in a graph or network format, and sequence diagrams, which illustrate interactions in a kind of fence format [6,15]. The use of collaboration diagrams for the expression of service interaction diagrams is preferred over the use of sequence diagrams, because collaboration diagrams are characterised by expressiveness,

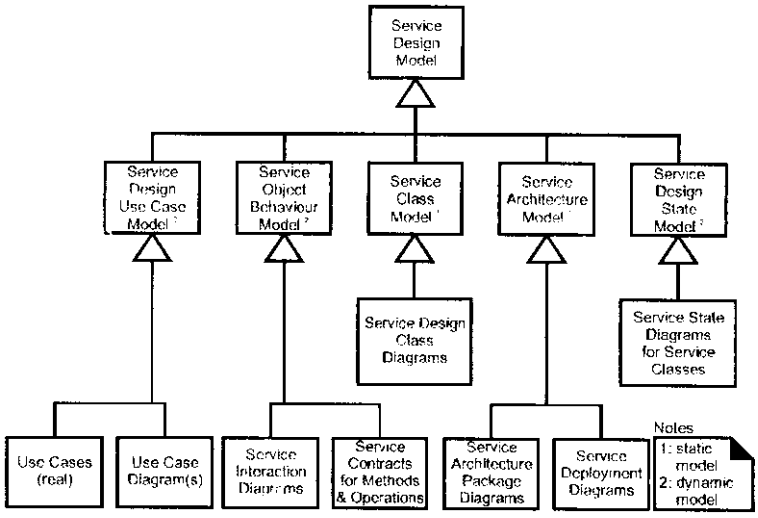

Fig. 13. The service design model.

an ability to convey more contextual information, and a relative spatial economy. Nevertheless, either notation can express similar constructs. What is really important is that service interaction diagrams is one of the most significant artifacts created during both service analysis and service design, because the skilful assignment of responsibilities to service objects and the design of collaborations between them are two of the most critical (for the satisfaction of the service requirements and thus for the successful realisation of a service) and unavoidable tasks (which also require the application of design skill) that have to be performed during service creation [21].

Another important artifact created during service design is the service design class diagram, which illustrates the specifications for the software classes of a telematic service using a strict and very informative notation. More specifically, from the service interaction diagrams the service designer identifies the software classes (service classes) that participate in the software realisation of the telematic service under examination, together with their methods, and from the service conceptual model the service designer adds detail to the service class definitions. A service design class diagram typically includes/illustrates service classes, their attributes and methods, attribute type information, navigability, and associations and dependencies between service classes. It has to be noted that in practice, service design class diagrams and service interaction diagrams are created usually in parallel. Furthermore, in contrast with a service conceptual model, a service design class diagram shows definitions of software entities (service components), rather than real-world concepts.

Service interaction diagrams and service design class diagrams belong to the service design model, which can be seen in Fig. 13. The service class model in this figure can be (optionally) further enhanced by the specification of service COs using the TINA-C Object Definition Language (ODL) [29], which is an enhancement (or a superset) of the Interface Definition Language (IDL) that has been introduced by the Object Management Group (OMG), and permits the definition of objects that have multiple interfaces and the definition of stream interfaces.

4.5. Service implementation

During this phase, an implementation of the telematic service (service code) is generated from the service specifications and the deployability of the overall implementation on a TINA-C compliant DPE is examined (DPE targeting). It is assumed that at the beginning of this phase, a specific (object-oriented) programming language and a specific distributed object platform are chosen.

The activities of the service implementation phase can be seen in Fig. 14. The dependencies between the artifacts produced during this phase can be seen in Fig. 15. This figure also shows the way that the service implementation phase artifacts depend on some of the service design phase artifacts.

The engineering representation of a service CO (using an object-oriented programming language like C++ or Java) is called an engineering Computational Object (eCO). The mapping between service COs and their eCOs is one to one; no eCO represents a composition of service COs nor is a service CO represented by more than one eCOs. The interfaces of an eCO represent the interfaces of its corresponding service CO [19]. However, these interfaces may be modified by type conversions for operation parameters or by adding operations necessary for the eCO's interaction with other engineering objects (e.g. those that provide required distribution transparencies). Furthermore, a management interface may be added, which includes operations to be performed by the object instance after its creation (constructor), just before its destruction (destructor), after activation or before deactivation. These modifications and additions

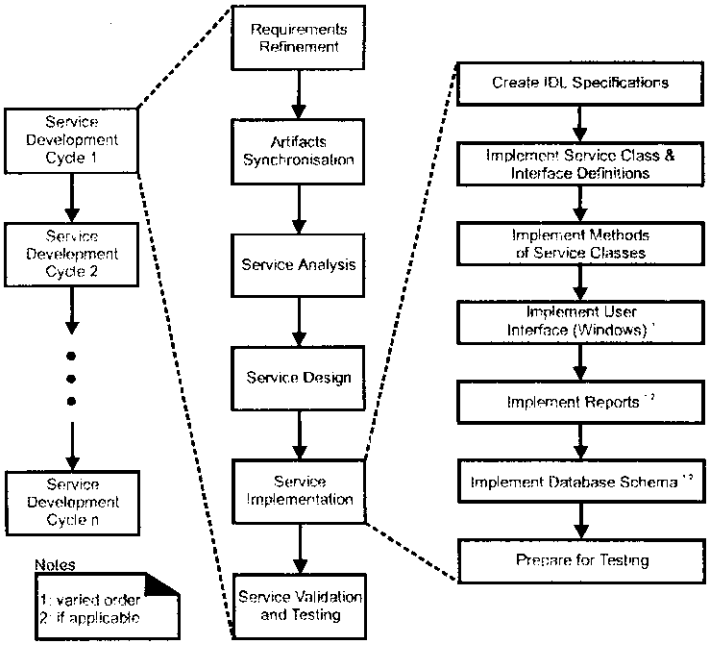

Fig. 14. Service implementation phase activities.

depend significantly on the exact characteristics of the selected DPE.

With the completion of the service design phase, there is sufficient detail to generate code for the service objects and construct the appropriate eCOs. For this purpose, the service design phase artifacts (and especially the service interaction diagrams and the service design class diagram) provide a significant degree of the necessary information and the translation process is relatively straightforward, especially if service classes are implemented (and tested) from the least coupled to the most coupled. More specifically, as a service interaction diagram shows the messages that are sent in response to a method invocation, the sequence of these messages translates to a series of statements in the method definition. Furthermore, from the service design class diagram, a mapping to the basic attribute definitions and method signatures is almost evident.

Despite these facts, the service implementation phase is not a trivial code generation process. The results generated during the service design phase have an approximate nature. During programming and testing, many changes will be made and detailed problems will be uncovered and resolved. However, because of the nature and structure of the proposed methodology, the service design phase artifacts will provide a resilient core that scales up with elegance and robustness to meet the new problems encountered during programming. Consequently, change and deviation from the service design phase artifacts during the service implementation phase should be expected and planned for (see also Section 4.6). After all, the spirit of iterative development is to capture a “reasonable” degree of information

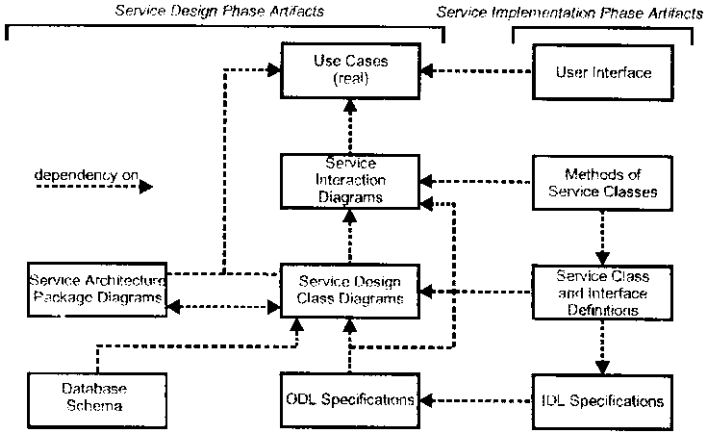

Fig. 15. Service implementation phase artifact dependencies.

during the service analysis phase, filling in details during the service design phase. Similarly, it is in the spirit of this process to capture a "reasonable" degree of design results during the service design phase, filling in further details during the service implementation phase. The definition of "reasonable" is, as it was expected, a matter of judgement [17,21].

Finally, the service implementation phase is a phase in which reusability matters should be considered seriously (i.e. more seriously than in the other phases). According to the TINA-C service architecture new services can be realised by enhancing already existing components (e.g. with the use of inheritance) or by defining new ones [27]. Therefore, the service independent components that are specified by TINA-C and other suitably created (and carefully selected) service components can be considered as reusable units in the creation of new services. They may be used in a service implementation as they are, or as the basis for the construction of a service specific component. More specifically, service dependent components may, either inherit or aggregate the characteristics of service independent components, or have a relation with them [4]. This activity can be facilitated greatly and enhanced by the construction of service independent component libraries, where components can be expressed in UML (various notations), ODL, C++/Java, etc. [11,30].

The exploitation of the available service independent components in the service implementation phase, and in previous phases (depending on the nature of the available component libraries), begins with the selection and reuse of the appropriate service independent functionality. Then, the service dependent segment is developed, by exploiting as much as possible the service independent segment. Finally, the two segments are integrated [1,13,25]. This process can be expressed with the following series of steps (fine tuning implies a feedback loop):

- Configure the access session related segment:
  - select the access session related functionality;
  - customise (if necessary) the selected access session related functionality.
- Configure the service session related segment:
  - select the service generic functionality;
  - customise (if necessary) the selected service generic functionality;
  - determine the service specific functionality;
  - develop the service specific functionality;
  - fine tune the relations between the service generic and the service specific part.
- Fine tune the relations between the access session and the service session segment.
- Configure the communication session related segment:
  - select the communication session related functionality;
  - customise (if necessary) the selected communication session related functionality.
- Fine tune the relations between the service session and the communication session segment.
- Integrate all three segments (access, service, and communication session).
- Prepare the end user system.

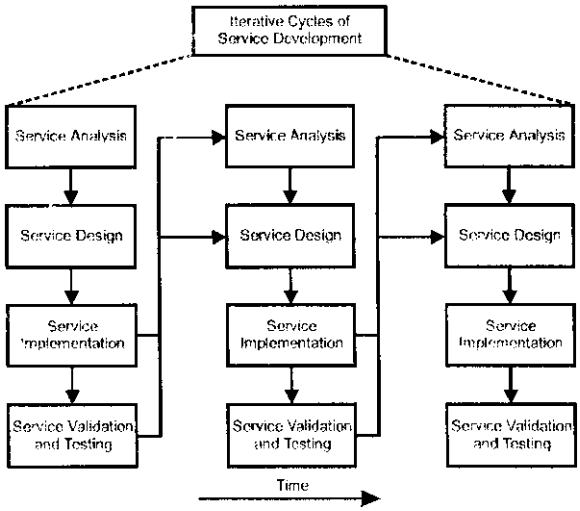

Fig. 16. The influence of service implementation work in the proposed iterative and incremental service development process.

4.6. Service validation and testing

Validation takes place in this phase by comparing the developed service software against the service specifications produced at the service design phase [2,16]. This activity can be subdivided into the following two subactivities.

- *Conformance testing*—it involves checking the implementation for conformance to architectural rules and standards used in the service design.
- *System testing*—it comprises the testing of service software in a (possible) operational environment.

With this phase a service development cycle ends and another one (depending on the exact nature of the specific service requirements) is ready to start. A significant strength of the iterative and incremental service development process adopted by the proposed methodology is that the results of a prior service development cycle can feed into the beginning of the next service development cycle. Thus, subsequent service analysis and service design results are being refined continually and informed from prior service implementation work (see Fig. 16). For example, when the code in cycle N deviates from the service design of cycle N (which it inevitably will), the final service design based on the implementation can be input into the service analysis and service design models of cycle N + 1. For this reason, as can be seen from Fig. 2, an early activity within a service development cycle is to synchronise the created artifacts. More specifically, the artifacts of cycle N will not match the final service code of cycle N, and they need to

be synchronised before extended with new service analysis and service design results.

5. Application of the methodology

In order to validate the proposed service development methodology and examine its usefulness, correctness, consistency, flexibility, and efficiency, several simple scenarios, regarding a variety of service creation activities for different simple telecommunications services, were considered. These scenarios confirmed that the methodology has all the above anticipated positive characteristics.

To verify and reinforce these findings under (more) realistic conditions, and to determine also the true practical value and applicability of the proposed methodology, all the phases of the methodology were used for the development of a real telematic service (a MultiMedia Conferencing Service for Education and Training, MMCS-ET) that is expected to have great demand in the near future [24]. This validation attempt, which provided valuable feedback and resulted in the further improvement of the methodology, is described briefly in this section, focusing mainly on the most important artifacts that were created during the application of the methodology. The intention is to provide characteristic examples on the use of the methodology to those interested in employing it for the development of new telecommunications services (e.g. service developers, service designers, etc.) and increase in that way their understanding of the methodology and their confidence on its effectiveness.

During the first phase of the methodology, it was found that the main requirement was to develop a telematic service (MMCS-ET) that will enable a teacher/trainer to teach efficiently and effectively a specific course to a number of geographically dispersed/distributed students/trainees. The service should establish an educational/training session between the teacher and the remote students that is equivalent to the educational/training session that would have been established between the same people (teacher and students) in a traditional classroom.

Further investigation revealed that every educational session has at least one teacher and that a teacher can participate in only one educational session at a given time, while a student can't participate in more than one educational sessions simultaneously. The establishment of an educational session involves the creation of the session by the teacher and the invitation of students to participate in the session. A student cannot create an educational session and cannot invite a new participant (teacher or student) to a session. A student participating in a session can only invite to direct communication with him/her another student, who is also (already) a participant in the same session.

The educational session established in a virtual classroom by the MMCS-ET should be equivalent to the educational session established in a traditional classroom and should have as many characteristics as possible in common with it. More specifically, in a traditional classroom the teacher has the ability to manage the educational session. Besides establishing the session he/she can also modify the session (e.g. by removing a student from the classroom and thus from the session), suspend and resume the session (e.g. by allowing the students to have a break for a few minutes without leaving the classroom), and finally finish the session (.e.g. by ringing a bell or by telling it to the students who leave the classroom and end the session). The same capabilities should also characterise the teacher in a virtual classroom.

Furthermore, in a traditional classroom the teacher and the students can interact (and possibly collaborate) during an educational session in the following ways:

- By seeing and talking to/hearing each other. This is the most common way of interaction.
- By writing at their notepads. In that way, a teacher can interact with only one student at a time, and e.g. see/correct his/her answer to an exam question.
- By writing at the blackboard of their classroom. In this case, everyone in the classroom sees what is written at the blackboard and everyone can write something at the blackboard.
- By exchanging course material (e.g. documents, graphs, etc.). This material can be used during or after the educational session, according (usually) to the instructions of the teacher.

Therefore, in a virtual classroom there is a need for audio/video (A/V) communication among all the session participants (to substitute face to face contact), text communication between only two session participants (as that achieved with the use of notepads), text communication among all the session participants (as that achieved with the use of a blackboard), file communication between the session participants (for the exchange of course material), and collaboration among all the session participants in order to perform a common task.

By analysing all the above mentioned (unstructured) requirements, a set of initial service requirements (categorised into session management, interaction, and collaboration support service requirements), together with a set of service functions were identified. From them the following use cases were deduced:

- Contact the MMCS-ET provider (start up the MMCS-ET service).
- Log in to the MMCS-ET provider domain.
- Start a new MMCS-ET session.
- Invite a student to join a MMCS-ET session.
- Join a MMCS-ET session after being invite.
- Invite a student (active user) to direct communication.
- Accept direct communication with a student (active user).

- Engage in text communication with an active user.
- Engage in file communication with an active user.
- Engage in A/V communication with a student (active user).
- Stop A/V communication with a student (active user).
- Start A/V communication with a student (active user).
- Terminate A/V communication with a student (active user).
- Engage in a chat with all active users.
- Engage in file communication with all students (active users).
- Start a voting process between all active users.
- Vote in a voting process between all active users.
- Present the outcome of a voting process to all the involved active users.
- Terminate direct communication between two students (active users).
- Remove a student from a MMCS-ET session.
- Terminate a MMCS-ET session.
- Terminate the MMCS-ET service.

These uses cases were considered in several iterative service development cycles. However, in this section, for reasons of clarity and simplicity, only one characteristic use case is examined (target use case) in the same way as if it was only one service development cycle. This use case is expressed in an expanded format in the following way:

Use case: invite a student to join an MMCS-ET session.
Actors: teacher (initiator), student.
Purpose: describes the way that a teacher invites a student to join an educational session.
Overview: a teacher (logged in user) invites a student (logged in user) to participate in an educational session. On completion, the student is left to decide whether to accept this invitation by the teacher or not.
Type: primary and essential.
Cross references: service Functions: SF.1.9., SF.1.10., SF.1.11., SF.1.13., SF.1.16.

Use cases—the users involved in the current use case must have completed the use case "Log in to the MMCS-ET Provider domain".

Typical course of events:

Actor action

(1) This use case begins when a teacher (logged in user) decides to invite a student (logged in user) to an educational session (a MMCS-ET session).

(2) The teacher specifies the user name of the student that he/she wants to invite to a MMCS-ET session and the name of the service (MMCS-ET) that the MMCS-ET session is part of.

Service Response

(3) Examines whether the teacher has started already a new MMCS-ET session or not, by querying the MMCS-ET provider profile.

If the teacher has not started already a new MMCS-ET session see/invite use case "Start a New MMCS-ET Session".

If the teacher has started already a new MMCS-ET session, continues.

(4) Locates a list containing information about all the users that participate in the MMCS-ET session (active users) with the help of the MMCS-ET session.

(5) Examines whether the user that initiated the invitation is active or not and whether he/she is a teacher or not, using the active user information list. Finds that the user that initiated the invitation is an active teacher.

(6) Locates the MMCS-ET service profile by querying the MMCS-ET session.

(7) Locates the user profile of the student that the teacher wants to invite (e.g. student A) by querying the MMCS-ET service profile.

(8) Informs the user profile of student A about the teacher invitation to join the MMCS-ET session.

(9) Examines whether the user that the teacher wants to invite is a student or not, by querying the appropriate user profile.

Finds that the user that the teacher wants to invite is a student.

(10) Examines the status of student A by querying the user profile of student A. The student can be, either a logged in user or an active user (participating already in a session of the MMCS-ET service or of another service).

Finds that student A is a logged in user.

(11) Locates a catalogue containing subscription information about all the users that are subscribers of the MMCS-ET service with the help of the user profile of student A.

(12) Examines whether student A is a subscriber of the MMCS-ET service or not, using the user subscription catalogue.

Finds that student A is a subscriber of the MMCS-ET service.

(13) Informs the MMCS-ET provider profile that student A has been invited by the teacher to join the MMCS-ET session.

(14) Prompts student A to accept or reject the invitation of the teacher to join the MMCS-ET session.

Alternative Courses:

- *Event 5:* The user that initiated the invitation is not active and/or he/she is not a teacher. However, only a teacher can invite a student to join a MMCS-ET session and a teacher after starting a new MMCS-ET session is always active. Indicate the error to the user.
- *Event 9:* The user that the teacher wants to invite is not a student (i.e. he/she is a teacher). However, only one teacher can participate in a MMCS-ET session at a given time. Indicate the error to the teacher.
- *Event 10:* Student A is an active user (i.e. he/she

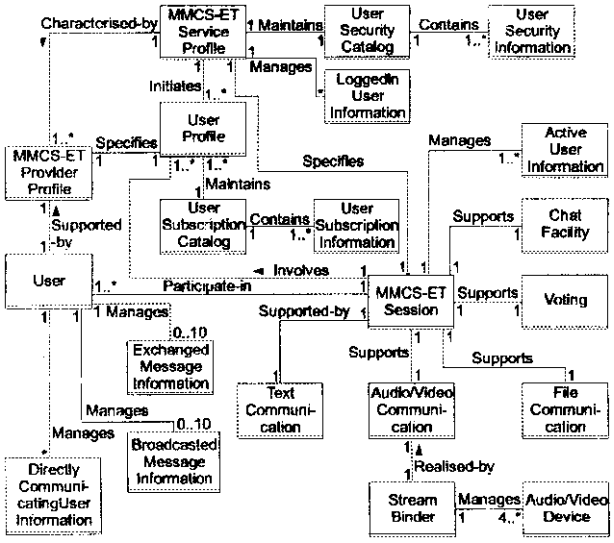

Fig. 17. The service conceptual model of the MMCS-ET.

participates already in a session of the MMCS-ET service or of another service). However, a student can participate in only one session of the MMCS-ET service or of another service at a given time. Indicate the error to the teacher.

- *Event 12:* Student A is not a subscriber of the MMCS-ET service. Indicate the error to the teacher.

The middle section of the expanded use cases (the "Typical Course of Events") is the most important as it describes in detail the required interaction between the actors and the telematic service. A critical aspect of this section is that it describes the most common sequence of activities needed for the successful completion of a service process. Alternative situations or exceptions that may arise with respect to the typical course were included in the final section of the expanded use cases (the "Alternative Course of Events").

Based on the expanded use cases, in the service analysis phase, the service conceptual model of Fig. 17 was created for the MMCS-ET. More specifically, the noun phrases (shown in italics) in the text of the expanded use cases were considered as candidate service concepts and attributes. Furthermore, as an attribute is a logical data value of a service object, the service conceptual model included all the attributes for which the service requirements suggested or implied a need to remember information. Service concepts are related by associations, which indicate some meaningful and interesting connection. Therefore, the service conceptual model of the MMCS-ET included all the associations for which knowledge of the corresponding relationship needs to be preserved for some duration ("need-to-know" associations). It has to be noted that it is generally undesirable to overwhelm the service conceptual model with associations that are not strongly required and that do not increase understanding.

In an attempt to gain an understanding of the service

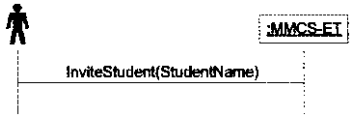

Fig. 18. Example of a service sequence diagram for the MMCS-ET.

behaviour, a service sequence diagram (see Fig. 18) was created for the typical course of events of each one of the identified use cases in the following way:

- a vertical line was drawn representing the MMCS-ET as a black box;
- each actor that directly operated on the MMCS-ET was identified and a vertical line was drawn for him/her;
- from the use case typical course of events text, the (external) service events that each actor generates were identified and illustrated in the correct order on the diagram.

The effect of the service operations that were revealed from the service sequence diagrams was described in service operation contracts. For each service operation, a service operation contract was constructed. Its "Responsibilities" section describes informally the purpose of the service operation, while its "Post-conditions" section describes declaratively the state changes that occur to service objects in the service conceptual model of Fig. 17, using a number of suitably selected statements (instance creation, instance deletion, attribute modification, association formed, association broken, and user interface activation). In full agreement, the service operation contract of the service operation suggested by the target use case, is the following:

Name: InviteStudent(StudentName:String).
Responsibilities: Invite a student (logged in user) to participate in an educational session (a MMCS-ET session).
Type: MMCS-ET.
Cross references: Service functions: SF.1.9., SF.1.10., SF.1.11., SF.1.13., SF.1.16.
Use case: "Invite a student to join a MMCS-ET session".
Notes:

- use a list containing information about all the users that participate in a MMCS-ET session (active users);
- use a catalogue containing subscription information about all the users that are subscribers of the MMCS-ET service.

Exceptions:
if the user that initiates the invitation is not active and/or he/she is not a teacher, indicate that it was an error;
if the user that the teacher wants to invite is not a student and/or he/she is already active, indicate that it was an error;
if student A is not a subscriber of the MMCS-ET service, indicate that it was an error.

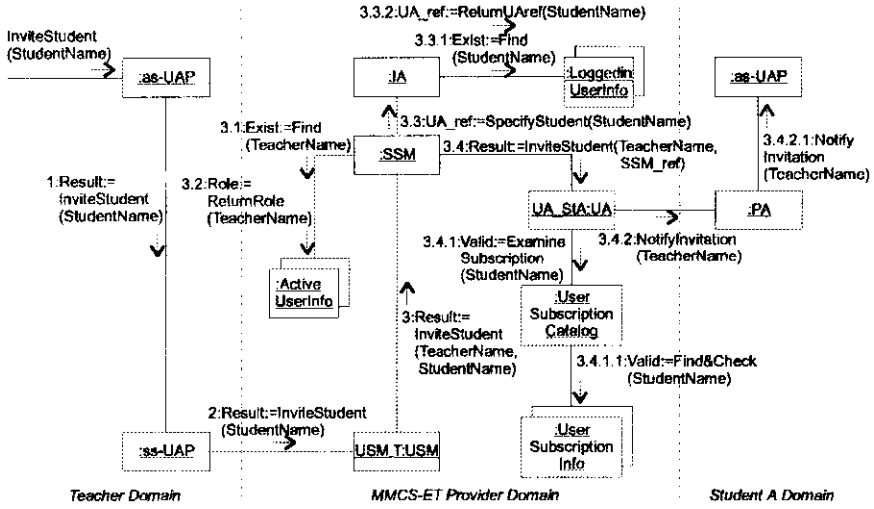

Fig. 19. Service interaction diagram for inviting a student to join a MMCS-ET session.

Output:

Pre-conditions:

- a teacher logged into the MMCS-ET provider domain.
- a student (e.g. student A) logged in to the MMCS-ET provider domain;
- the teacher decided to invite student A to a MMCS-ET session;
- the teacher specified the user name of student A.

Post-conditions:

- The*User* (corresponding to the teacher) was associated with the *MMCS-ETProviderProfile* (*association formed*).
- This *User* was associated with the *MMCS-ETSession*, based on the confirmation that *User SessionCreation* was set to true (*association formed*).
- The *MMCS-ETSession* was associated with *ActiveUserInformation*, based on the satisfaction of the following conditions (*association formed*).
  - The user that initiated the invitation was active.
  - The user that initiated the invitation was a teacher.
- The MMCS-ETSession was associated with the MMCS-ETServiceProfile (association formed).
- The MMCS-ETServiceProfile was associated with LoggedinUserInformation regarding student A (association formed).
- The *MMCS-ETServiceProfile* was associated with the *UserProfile* (corresponding to student A) (*association formed*).
- This *UserProfile* was associated with the *UserSubscriptionCatalog*, based on the satisfaction of the following conditions (*association formed*):
  - *UserProfile.Role* was set already to student.
  - *UserProfile.Status* was set already to logged in.
- The *UserSubscriptionCatalog* was associated with *UserSubscriptionInformation*, based on the confirmation that student A was (at that time) a subscriber of the MMCS-ET service (*association formed*).
- The *UserProfile*(corresponding to student A) was associated with the *MMCS-ETProvider Profile* (association formed).
- The *MMCS-ETProviderProfile* was associated with the *User* (corresponding to student A) (*association formed*).
- Interaction with student A was prepared (*user interface activation*).

Taking into account all the artifacts produced so far, in the service design phase, a service interaction diagram in the form of a UML collaboration diagram was created for each one of the identified service operations. The objective was to fulfil the post-conditions of the corresponding service operation contracts, recognising however, that the previously defined post-conditions are merely an initial best guess or estimate of what must be achieved, and therefore their accuracy should be questioned.

From these service interaction diagrams (an example of which is depicted in Fig. 19 for target use case) the way that the MMCS-ET service COs communicate via messages in order to fulfil the service requirements is evident. The participating MMCS-ET service objects were drawn from the service conceptual model of Fig. 17, after taking into account the service components proposed by the TINA-C service architecture. Therefore, the links between the MMCS-ET service objects are actually instances of the associations present in the service conceptual model of Fig. 17, after taking into account the service components proposed by the TINA-C service architecture. Therefore, the links between the MMCS-ET service objects are actually instances of the associations present in the service conceptual model of Fig. 17, represent connection paths between service object instances, and indicate that some form of navigation and visibility between the instances is possible (attribute, parameter, locally declared or global visibility). Finally, it has to be noted that in order to

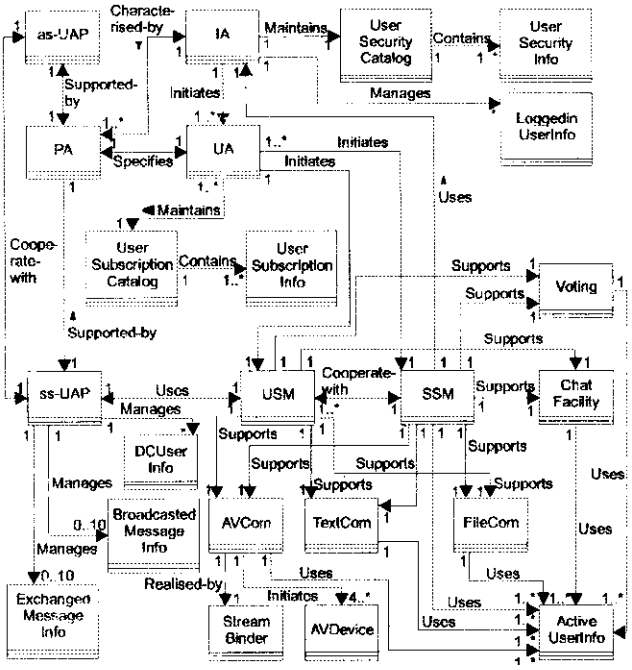

Fig. 20. The service design class diagram of the MMCS-ET (simplified).

illustrate the creation (or deletion) of service object instances (from service classes), a language-independent creation (or deletion) message (create or delete) is shown being sent to the instance being created (or deleted).

By analysing the service interaction diagrams, all the service classes (together with their attributes and methods) participating in the software realisation of the MMCS-ET were identified and illustrated (with simplifications) in the service design class diagram of Fig. 20. The associations present in this diagram satisfy the ongoing "memory needs" revealed by the service interaction diagrams and the navigability arrows on them indicate the direction of attribute visibility (non-attribute visibility is indicated by dependency relationships). Creation (deletion) -related methods were omitted from the service design class diagram, because they can have multiple (implementation specific) interpretations, and because they represent a very common activity. For similar reasons, accessing methods (those which retrieve or set attributes) were also excluded from depiction in the service design class diagram, in order to keep it concise and focused.

Considering all the artifacts produced in the service design phase, the MMCS-ET was implemented using Microsoft's Visual C++ (ver. 6.0) together with Microsoft's Distributed Component Object Model (DCOM) [7] (appropriately extended with a high-level API in order to support continuous media interactions) on MS Windows NT 4.0, and was executed on a number of workstations connected via a 10 Mbit/s Ethernet LAN [4,24]. All the interconnected workstations belong to the same (MS Windows NT) domain and one of them functions as a primary domain controller.

6. Exploitation of design patterns and frameworks

A typical telematic service is a large scale system as it consists of thousands of software objects that are running on hundreds of hardware objects, utilising a great variety of network resources, and interacting in a complex and almost unpredictable way [3]. For this reason, the successful application of the proposed service creation methodology can be a difficult task, where the effective and efficient communication of architectural knowledge between the service designers and developers is of great importance. To facilitate this communication the exploitation of design patterns and frameworks in the service engineering area is suggested.

A framework can be defined as a set of prefabricated services together with some architectural concepts that define the constraints to put these services together. The architecture includes the rules that can be used to integrate the single services and to define possible flows of control between them [15]. Design patterns represent abstract solutions for specific problem classes. They capture the static and dynamic structure, and collaboration of a group of objects. They can be defined in an abstract, language independent way [17]. While design patterns can be considered as a horizontal structure over a set of COs, frameworks can be considered as vertical, domain specific (e.g. telecommunications specific) configurations of components.

In the case of TINA-C, design patterns can be defined by identifying groups of interworking service objects, where every group is characterised by a micro-architecture that determines the way the objects interact to provide a solution for the specific aspects of a subproblem that arises during the development of a telematic service. Furthermore, a framework can be defined as the overall architecture, which specifies how the identified configurations of service objects can collaborate to implement a solution for the whole problem. Thus, a framework is a kind of construction kit for complete or semi-complete telematic services. It has to be complemented and customised using inheritance techniques [13]. As an example, the TINA-C service architecture can be defined as a framework. On the contrary, the access to the usage of a service is an example of a design pattern (access pattern). It specifies the object group of the access session.

The introduction of design patterns and frameworks in the proposed methodology implies the establishment of a common vocabulary and the definition of common design structures for all persons involved. They assist to reduce the scope of the problem solving process in the case of service creation, because they support the identification of similar problems and similar solutions. However, design patterns and frameworks are abstract concepts. There is no guarantee that their usage will lead to design reusability, design portability, and abstract customisability. Furthermore, good design patterns and frameworks, like good inheritance hierarchies, can not be invented in an easy way. They have to be

chosen and designed very carefully [20]. Otherwise, the introduction of insufficient and wrong chosen patterns and frameworks in the service creation process may hinder or even prevent the design and implementation of successful telematic services.

For the definition and enhancement of service engineering design patterns and frameworks a step by step "case study" approach is proposed. Initially, the main design patterns and the related core classes are identified, and form conceptually a framework. Then, a first prototype of the framework is created. After testing and evaluating this prototype, the framework can be extended with respect to the results of the evaluation process and additional case studies. During these extensions further aspects, which are candidates for the definition of new design patterns and abstract classes can be identified and integrated into the framework [13,17].

7. Conclusions

A revolution in information technology and telecommunications is already in progress, and is expected to escalate rapidly in the near future. The two worlds have already begun to converge and the convergence path is marked by the continuously expanding penetration and scope of telecommunications services [5]. Considering this evolution, emerging telecommunications systems promise to offer a wide variety of highly sophisticated, personalised, affordable, high quality, and ubiquitous services over the widest possible coverage area. Several providers are involved in such an ambitious (yet realistic) service provision scenario in which competition will mostly focus at the service level, with multiple providers offering new services to the market in a short time over a variety of networks and end systems [12,23]. In the light of these challenges and because of the highly increasing complexity of new telecommunications services and the inherent distributed nature of them, a methodology covering the whole service development process, like the one proposed and examined in this paper, is absolutely necessary.

This methodology enforces the service developer to take into account all the necessary features for the successful design and realisation of a service by exploiting the session-oriented, multi-party, multi-domain nature of the TINA-C service architecture. It provides a step by step approach from problem definition to the realisation of new telecommunications services. For this purpose, the service is studied and described (in a number of service development cycles) at hierarchically related abstraction levels, in the sense that at each level the results achieved at previous levels of abstraction are preserved and refined. Additionally, the use of the object-oriented paradigm is advocated all along the development process. Consequently, this methodology combines the benefits of object-oriented modelling, particularly in terms of scalability and reusability, with those provided by a top–down approach [11,16]. Moreover, as service requirements are emphasized, such an approach ensures a high-level of confidence that the users' expectations on the service will be met.

The application of the proposed service creation methodology to the development of the MMCS-ET has enlightened many aspects regarding its structure and its use, and offered confidence that it can enable the fast and efficient creation of telematic services. It must be kept in mind, however, that to obtain the maximum possible productivity gains and to exploit the full potential of the methodology it is not sufficient to apply it in a mechanical manner. On the contrary, an adaptation of the methodology to the service developer's attitude and to the wider organisational mentality and approach regarding telecommunications and information technology in general, is required. In that way, the proposed methodology will be able to support service creation activities even more effectively, without restricting the creativity of service developers, and by utilising fully their prior service development experience.

References

[1] ACTS Project AC227 (SCREEN), Basic Object-Oriented Technology for Service Creation, CEC Deliverable D21, 1996.
[2] D.X. Adamopoulos, G. Haramis, C.A. Papandreou, Rapid prototyping of new telecommunications services: a procedural approach, Computer Communications 21 (1998) 211–219.
[3] D.X. Adamopoulos, C.A. Papandreou, An integrated object-oriented approach to telecommunications service engineering, Proceedings of IFAC/IFOR/IMACS/IFIP LSS '98, Rio, Greece, 1998, pp. 834–839.
[4] D.X. Adamopoulos, G. Pavlou, C. Papandreou, Supporting advanced multimedia telecommunications services using the distributed component object model, Proceedings of IS and N 2000, Lecture Notes in Computer Science, vol. 1774, Springer, Berlin, 2000, pp. 89–104.
[5] H. Berndt, T. Hamada, P. Graubmann, TINA: its achievements and its future directions, IEEE Communications Surveys & Tutorials, vol. 3, no. 1, First Quarter, 2000.
[6] G. Booch, J. Rumbaugh, I. Jacobson, Unified Modelling Language User Guide, ACM Press, New York, 1998.
[7] N. Brown, C. Kindel, Distributed Component Object Model Protocol-DCOM, Microsoft Corporation, January 1998.
[8] B. Dano, H. Briand, F. Barbier, A use case driven requirements engineering process, Requirements Engineering 2 (1997) 79–91.
[9] L.A. De la Fuente, L. Ferrari, J. Gallego, P. Llamas, The Eurescom P610 project: providing framework, architecture and methodology for multimedia services management, Proceedings of IFIP/IEEE DSOM '97, Sydney, Australia, 1997, pp. 145–154.
[10] M. Declan, Adopting object oriented analysis for telecommunications systems development, Proceedings of IS and N '97, Lecture Notes in Computer Science, Springer, Berlin, vol. 1238, 1997, pp. 117–125.
[11] P.P. Demestichas, N.P. Polydorou, A.K. Kaltabani, N.I. Liossis, S. Kotrotsos, E.C. Tzifa, M.E. Anagnostou, Issues in service creation for open distributed processing environments, Proceedings of ICC'99, June 1999.
[12] T.M. Didriksen, L.S. Sorumgard, T.O. Molnes, Inexpensive open distributed service platform, Proceedings of IFIP SmartNet'99, Thailand, November 1999.
[13] K.P. Eckert, P. Schoo, Engineering frameworks: a prerequisite for the design and implementation of distributed enterprise objects, Proceedings of EDOC'97, October 1997, pp. 170–181.

[14] S. Efremidis, D. Prevedourou, L. Demounem, K. Milsted, H. Zuidweg, TINA-oriented service engineering support to service composition and federation, Proceedings of IS and N '98, Lecture Notes in Computer Science, Springer, Berlin, vol. 1430, 1998, pp. 409–422.
[15] P. Evits, A UML pattern language, Macmillan Technology Series, February 2000.
[16] J.P. Gaspoz, Methodology for the development of distributed telecommunications services, Journal of Systems and Software (1996) 1–22.
[17] T. Hansen, Development of successful object-oriented frameworks, Proceedings of ACM SIGPLAN OOPSLA '97, Atlanta, USA, 1997, pp. 115–119.
[18] M.M. Kandé, S. Mazaher, O. Prnjat, L. Sacks, M. Wittig, Applying UML to design an inter-domain service management application, Proceedings of UML '98, 1998, pp 1–9.
[19] E. Kelly, N. Mercouroff, P. Graubmann, TINA-C DPE architecture and tools, Proceedings of TINA'95, February 1995, pp. 39–54.
[20] E. Koerner, Patterns for constructing CSCW applications in TINA, Proceedings of IDMS'97, Lecture Notes in Computer Science, Springer, Berlin, vol. 1309, 1997, pp.322–329.
[21] C. Larman, Applying UML and patterns: an introduction to object-oriented analysis and design, Prentice-Hall, Englewood Cliffs, NJ, 1998.
[22] D. Lewis, T. Tiropanis, Integrating TINA into an internet-based services market, Proceedings of IS and N'98, Lecture Notes in Computer Science, Springer, Berlin, vol. 1430, 1998, pp. 183–191.
[23] T. Mota, P. Hellemans, T. Tiropanis, TINA as a virtual market place for telecommunication and information services: the VITAL experiment, Proceedings of TINA '99, April 1999.
[24] C.A. Papandreou, D.X. Adamopoulos, Design of an Interactive Teletraining System, BT Engineering 17 (1998) 175–181.
[25] N.D. Polydorou, N.I. Liossis, E.C. Tzifa, P.P. Demestichas, M.E. Anagnostou, Efficient creation and development of telecommunication services in heterogeneous distributed processing environments, Proceedings of IEEE/IEE ICT '98, IV, 1998, pp. 336–340.
[26] S. Rana, E. Sellin, Implementation of a pan-European TINA-compliant service management platform, Computing and Control Engineering Journal 10 (1999) 73–78.
[27] TINA-C, Definition of Service Architecture, Version 5.0, 1997.
[28] TINA-C, TINA Business Model and Reference Points 4.0, 1997.
[29] TINA-C, TINA Object Definition Language, TR_NM_.002_2.2_9.6, 1996.
[30] K. Verschaeve, B. Wydaeghe, F. Westerhuis, J. De Moerloose, Multilevel component oriented methodology for service creation, Proceedings of IS and N 2000, Lecture Notes in Computer Science, Springer, Berlin, vol. 1774, 2000, pp. 169–179.