ACKNOWLEDGEMENTS

I would like to thank the members of the work package one of the PHOSPHORUS project by their contributions (Angel Sánchez and Joan Antoni Garcia), to Fernando Agraz and especially to Eduard Grasa and Sergi Figuerola, who have guided me into the development process of this project.

I also would like to thank my family and my closest friends for the moral support they have given me during all the time this project has last.

INDEX

INDEX OF FIGURES

INDEX OF TABLES

1. INTRODUCTION

1.1. ORGANIZATION OF THE DOCUMENTATION

This documentation corresponds to the thesis of the project "Design and Implementation of a Network Resources Reservation System". This project implements a service called ARS (Advance Reservation Service) that integrates two international projects.

The development of this project has followed the steps of UP (Unified Process) methodology. The main steps of this methodology are the following ones:

- Planning and elaboration: planning the phases of the project, define the glossary, the use cases, the architecture of the system, the requirements, etc.
- Construction: development of the system. The specification and design is done, and for each phase of the project the use cases, the glossary, the state diagrams, the sequence diagrams, the operations, etc. are redefined if it is necessary.
- Implementation: implement the system and testing it.

It has been decided to use this type of methodology, a traditional methodology instead of an agile one [27], because this project is part of a two international projects, as it will be explained later, and to coordinate all the parts of both projects it is better to use this type of methodology.

The documentation of this thesis is structured as the corresponding steps named before. Thus, this documentation is organized as follows.

In the second chapter, the environment where the project has been developed, a brief description of the work done and the main goals is done.

The third chapter presents the predecessors and the feasibility of the project developed.

In chapter number four, the functionalities of the system are described in detail. This chapter is done following the steps of the Volere methodology in order to make deeply the requirements analysis. This part of the project is written in the present tense because the requirements analysis is done at the beginning of the project and it is modified during the development because new requirements may appear.

The fifth chapter contains the planning of the project. It is explained the initial planning and the final one, because the initial planning has been modified, as it will be explained.

The sixth chapter shows the economic analysis done for this project.

The chapters 7, 8, 9 and 10 are the main part of the development process. They contain the specification, design, implementation and testing of the system.

In the chapter number eleven the demonstrations where the ARS has been involved are explained.

In chapter 12 the future work is presented. It talks about the work that should be done as a follow up of this project.

The last chapter presents the conclusions obtained during the degree, during the development of the project and once the project has been finished.

Finally, a glossary with the important concepts, the acknowledgements and the references are presented.

2. CONTEXT OF THE PROJECT

2.1. INTRODUCTION

This project implements an Advance Reservation Service (ARS) that integrates the UCLP (User Controlled Lightpath Provisioning) system with the European project PHOSPHORUS IST FP6. The main goal of this service is to provide the UCLP system with the capability to make advance reservations of the network resources. In this way, UCLP will permit its users to make advance reservations and also will be a part of the PHOSPHORUS IST FP6 project. In other words, networks managed by UCLP will be able to be interconnected with other networks managed by the different systems integrated in the PHOSPHORUS IST FP6 project.

To sum up, the ARS allows the UCLP system to support advance reservations and integrates this one with the PHOSPHORUS IST PF6 project. In this way, advance reservations through networks managed by the UCLP system and by the other systems integrated with the PHOSPHORUS IST FP6 project can be done at the same time.

Following there is a little introduction to the UCLP system and the PHOSPHORUS IST PF6 project.

2.1.1. UCLP

The UCLP [7] system was born as a result of a RFP (Request For Proposals) issued by CANARIE in September 2003. Four implementation proposals prepared by four independent teams were accepted, among them the proposal of the CRC (Communications Research Centre, Canada) and the UofO (University of Ottawa, Canada). The i2cat foundation joined the team in April 2004.

UCLP software allows end-users, either people or sophisticated applications, to treat network resources as software objects and provision and reconfigure lightpaths within a single domain or across multiple, independently managed, domains. Users can also join or divide lightpaths and hand off control and management of these larger or smaller private sub-networks to other users.

The UCLP software is designed to enable end-users to create their own discipline or application-specific IP network, particularly in support of high-end e-science and Grid applications. In other words, UCLP can be very simply thought of as a configuration and partition manager that exposes each lightpath in a physical network and each network element associated with a lightpath as an ‘object’ or ‘service’ that can be put under the control of different network users to create their own IP network topologies.

This way several network operators can make part of their resources available to end users so that they can decide when they want to create/delete end to end connections or change the network topology (that’s why the system is called User Controlled Lighpath Provisioning).

I. UCLP Objectives

UCLP was born to satisfy the requirements of certain number of applications that cannot be implemented using centralized network management tools. Some of these applications are:

- **Customer controlled and managed networks**. Universities, regional networks, governments and large enterprises are building their own

communication networks under condominium agreements. Participants in such agreements join efforts to buy together physical infrastructures. Each institution gets a part of the resources proportional to its initial investment. Each institution wants to manage his resources independently from all the others, having the capability of implementing its own topology discover or protection mechanisms and being able to offer its customers added value services.

- **Dedicated IP networks**. Current IP networks are optimized for thousands of clients with relatively small traffic flows. All over the world are emerging small communities who need to interchange high amounts of traffic at a high big rate (grid applications, e-science, sensor and instrument networks). These communities cannot use the public Internet to carry out their experiments; they need dedicated IP networks where routers and computers where high-end applications are running have dedicated end to end links.
- **User controlled traffic engineering**. UCLP allows local and/or regional network managers to apply their own traffic engineering policies. Using UCLP they can create new BGP paths between two networks creating a direct optical connection. Big companies and regional networks can be interconnected directly using a peering point instead of a hierarchical IP network.

II. Parallel Virtual Networks

Using UCLP, several network operators can make part of their resources available to end users so that they can assemble these various web services into *multiple network virtualizations, or Articulated Private Networks (APNs)* running their own protocols and services. UCLP also allows users to "spawn" new virtualized networks, at any time, without requiring setup or permission of any central operator. This will allow for the easy migration of new services and protocols. This capability is already available on CAnet4 today with over 30 separate virtualized networks or APNs used for various network research and cyber-infrastructure applications. It also allows researchers to carry out experimentation in various new services such as security, active networking, etc.

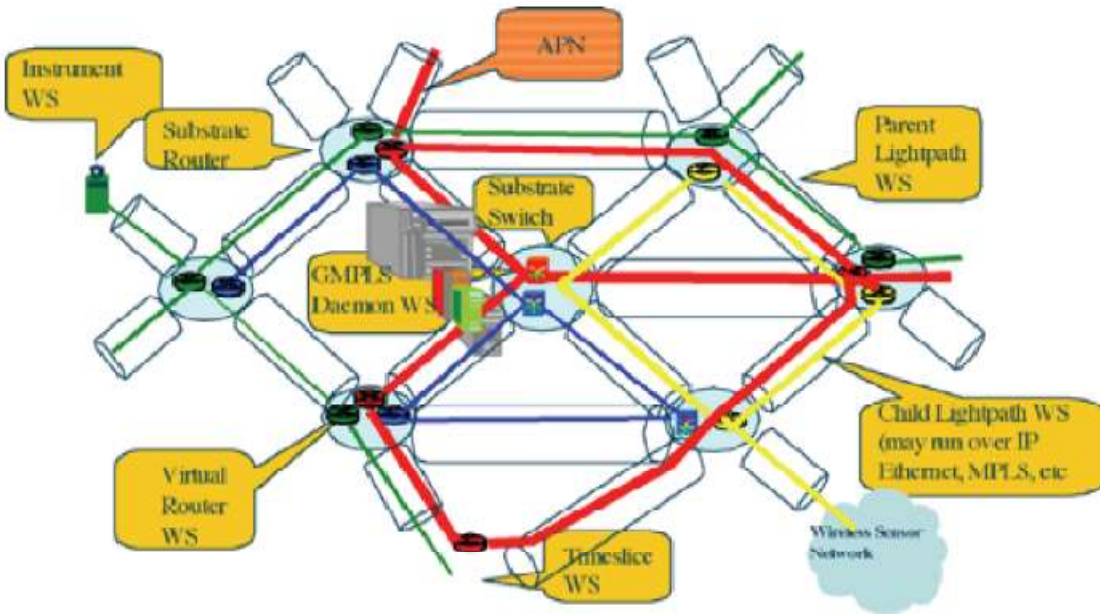


Figure 1. Parallel Virtual Networks

III. Extending the Application into the Network

The virtualized network resources can also be linked together with other network resources, instruments, sensors, etc. via workflow languages such as BPEL to create new rich web services. This opens the door for users to create any number of workflows that can make use of the network resources whenever they are required, effectively bringing the network into the application itself. Not only can the end users control and manage their own private high speed networks, but also can their sensors, instruments and applications leaving them free to analyze the data that is being transferred instead of having to worry about how to get the data in the first place.

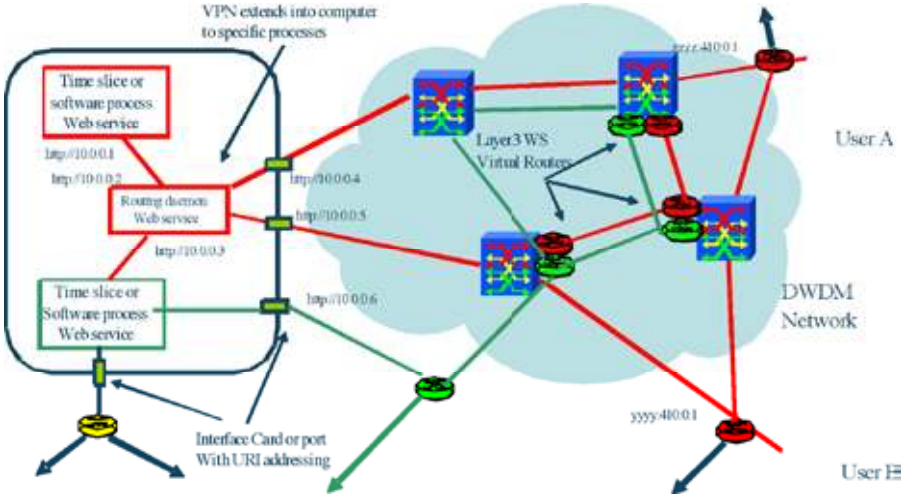


Figure 2. Extending the application into the network

IV. Articulated Private Networks in UCLP

The UCLP software provides an alternate network provisioning process which allows users to control their own packet or switched based network architecture including topology, routing, virtual routers, switches, virtual machines and protocols. The UCLP concept consists of many separate, concurrent and independently managed Articulated Private Networks (APNs) operating on top of one or more network substrates across different ownership domains. An APN can be considered as physically isolated or "underlay" network where a user can create a customized multi-domain network topology by binding together layer 1 through 3 network links, computers, time slices and virtual or real routing and/or switching nodes. This UCLP capability is realized by representing all network elements, devices and links as web services, and by using web services workflow to allow the user to bind together their various web services to create a long lived APN instantiation.

The current UCLP can also be used for bandwidth on demand (by adding a bandwidth on demand web service on top of the APN), but its main feature is that it allows users to gain a certain degree of control over parts of real network devices (called network resources). This way a user can gather network resources from different physical network providers and create its own multidomain physical network, called APN in the UCLP terminology. The user is the administrator of this APN, therefore he can decide what types of services he wants to deploy on top of this APN, like a bandwidth on demand service, an advance reservations service, the APN Scenarios Service (explained later in this paper) or any other service he decides. The user may also choose not to deploy any service and use the APN for his own needs, through the use of the UCLP Management Centre (the Graphical User Interface).

The following figure provides a high-level overview of the different web services that compose the core of the UCLP software.

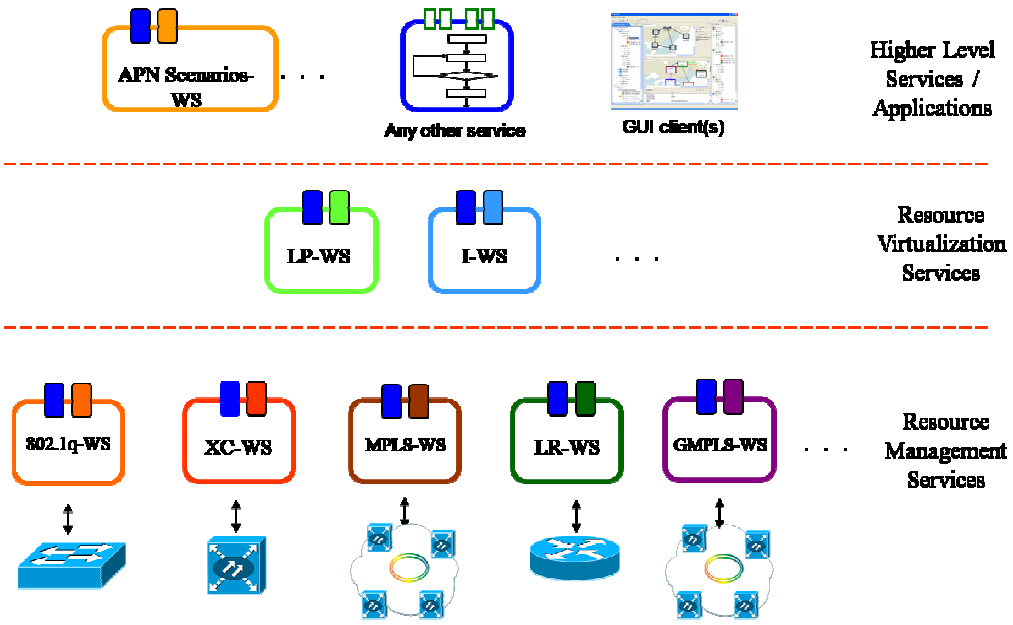


Figure 3. UCLP Service Oriented Architecture

2.1.2. PHOSPHORUS IST FP6

PHOSPHORUS IST FP6 [8] is an European project that focuses on delivering advances network services to Grid users and applications interconnected by heterogeneous infrastructures. The project addresses some of the key technical challenges to enable on-demand end-to-end network services across multiple domains. The Phosphorus network concept and test-bed make applications aware of their Grid resource environment (computational and networking) and its capabilities. Phosphorus enables and tests dynamic, adaptive and optimised used of heterogeneous network infrastructures interconnecting various high-end resources.

I. PHOSPHORUS objectives

The main objectives of the project are the following ones:

- **Delivery of single-step on-demand services across multi-domain networks for e-science applications.** The project aims to demonstrate on-demand service delivery across access-independent multi-domain/multi-vendor research network test-beds on European and worldwide scale. The global test-bed in Phosphorus project will be composed of a number of local test-beds interconnected using multiple optical international networks. These will include GÉANT2, CBDF, GLIF connections and NRENs (National Research and Education Networks). It is also important to note the existence of different management systems such as: GMPLS, UCLP, DRAC [12] and ARGON [11].
- **Seamless way for Grid systems to access network resources and Grid middleware extensions to GMPLS protocol.** The goal is to develop integration between applications, middleware and transport networks, based on three planes: Network Service Plane (NSP), Network Resources Provisioning Systems (NRPS) plane and control plane. Service plane will consist of APIs

specification for applications, services components exposing network and Grid resources in integrated fashion taking into account policy driven AAA mechanisms. Construction of NRPS plane assumes adaptation of existing NRPSes and full integration with middleware and control plane. GMPLS control plane will be enriched with Grid extensions providing Grid middleware with access to optical network resources as first-class Grid resources.

- **Conduct accompanying studies to investigate and evaluate the further technological development of the project outcomes.** Supporting studies will be carried out throughout the project in order to ensure a proper future outlook for the project results. Resource management and job scheduling algorithms will be studied, designed and finally tested in a simulation environment developed by the project. They will incorporate issues of network awareness, constraint based routing and advance reservation techniques. Recommendation for the design of an optical control plane will be analyzed and documented.

II. NRPSs involved in Phosphorus

ARGON (Allocation and Reservation in Grid-enabled Optic Networks): was developed to manage resources of advanced network equipment as its present in the German VIOLA testbed. The advance reservation service of ARGON is able to operate on the GMPLS as well as on the MPLS level. It guarantees the requested level of QoS for applications for the requested time interval. This feature enables a Meta-Scheduling Service to seamlessly integrate the network resources into a Grid environment.

Nortel's DRAC (Dynamic Resource Allocation Controller): is the world's first commercial-grade network abstraction and mediation middleware platform, acting as an agent for network clients (users, applications, compute resource managers) to negotiate and reserve appropriate network resources on their behalf. DRAC uses client's QoS requirements and pre-defined policies to negotiate end-to-end connectivity across heterogeneous domains in support of just-in-time or scheduled computing workflows.

UCLP: see section 2.1.1.

III. System architecture

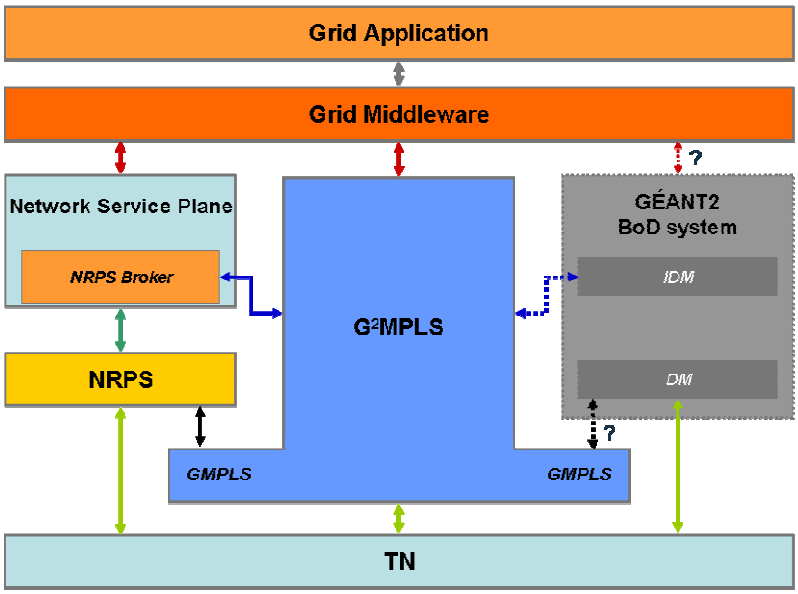


Figure 4. PHOSPHORUS architecture

The Grid Layer will be based on the Globus Toolkit 4 [15] and UNICORE [18] software. It is responsible for running a set of applications; it also forwards to the Network Service Plane the network requirements to run these applications. The Grid Layer is also referred as Grid Middleware.

The NSP is the adaptation layer between the Grid layer and the NRPS layer. It coordinates the different networks in the NRPS layer to meet network resources requests coming from the Grid Layer. This block is a key element to provide interoperability between the different NRPSs, GMPLS and the Grid Layer.

The NRPS Layer is composed by several Network Resource Provisioning Systems managing different domains. For the scope of the Phosphorus project these NRPSs can be, as seen before, either ARGON, DRAC or UCLP system. Each system is responsible for the management of the resources available in its domain. This management can be achieved by directly accessing the nodes of the transport network, or, if available, by means of a GMPLS control plane that configures the network.

An NRPS Adapter is needed because each NRPS has a particular interface an adapter between the NRPSs and the NSP. The NRPS Adapter is the responsible for the mapping of these interfaces to a common interface.

The Transport Network (TN) is a group of physical devices and links managed by an entity (an NRPS or a GMPLS CP) that allows the transport of information between endpoints.

G2MPLS is an enhancement of the ASON/GMPLS Control Plane architecture that implements the concept of Grid Network Services (GNS). In the PHOSPHORUS framework, GNS is a service that allows the provisioning of network and Grid resources in a single-step, through a set of seamlessly integrated procedures.

IV. Project Structure

The project is divided in the following work packages:

- WP0 – Project Management
- WP1 – Network Resource Provisioning Systems (NRPS) for Grid Network Services (GNS)
- WP2 – Enhancements to the GMPLS Control Plane for Grid Network Services (GNS)
- WP3 – Middleware and applications
- WP4 – Authentication, Authorization and Accounting
- WP5 – Supporting studies
- WP6 – Test-bed and demonstration activities
- WP7 – Dissemination, Contributions to standards, Liaisons

In the following figure, the structure of the project and how are placed the packages can be seen.

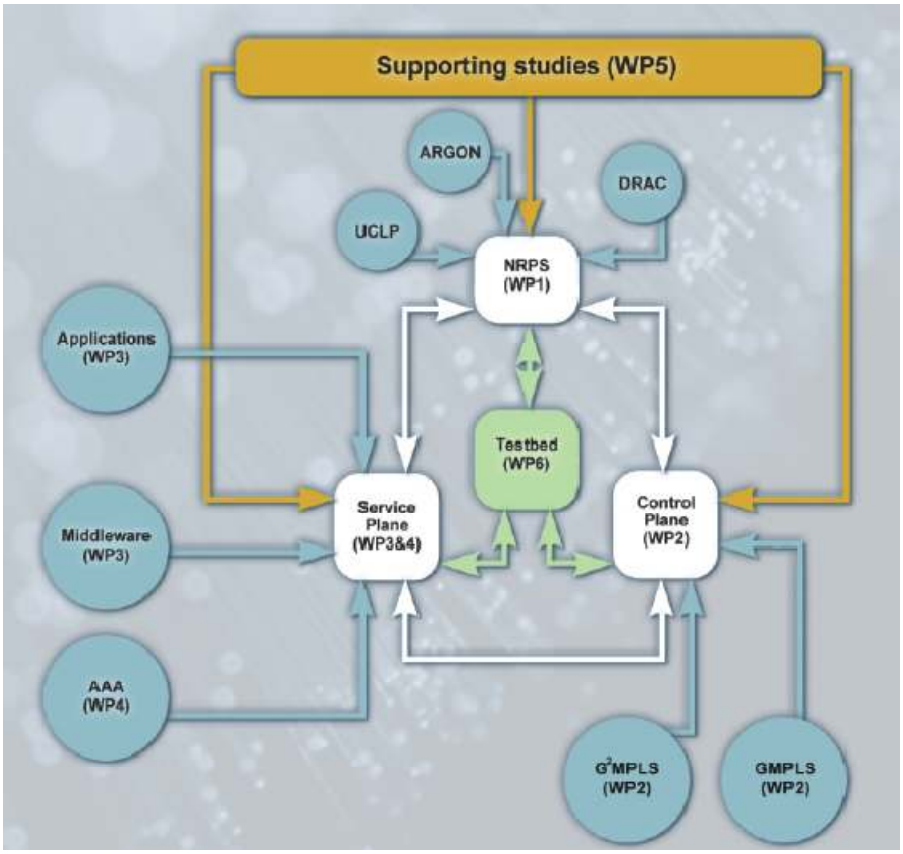


Figure 5. Project structure

2.2. General Description of the ARS

As the PHOSPHORUS IST FP6 project is divided in seven packages, the integration of the UCLP into this one is made through the work package one (WP1), that treats about the requirements and the specifications of the interfaces and the architecture for the interoperability between NRPS, GMPLS, and the Middleware.

As can be seen in figure number 6, the integration of the UCLP system into the PHOSPHORUS IST PF6 project using the ARS is done by means of an interface that contains all the functionalities of the ARS (this interface will be explained in detail in the section 9). When any of these functionalities is called, the ARS processes the request, computes the results and then, returns the response to the requestor.

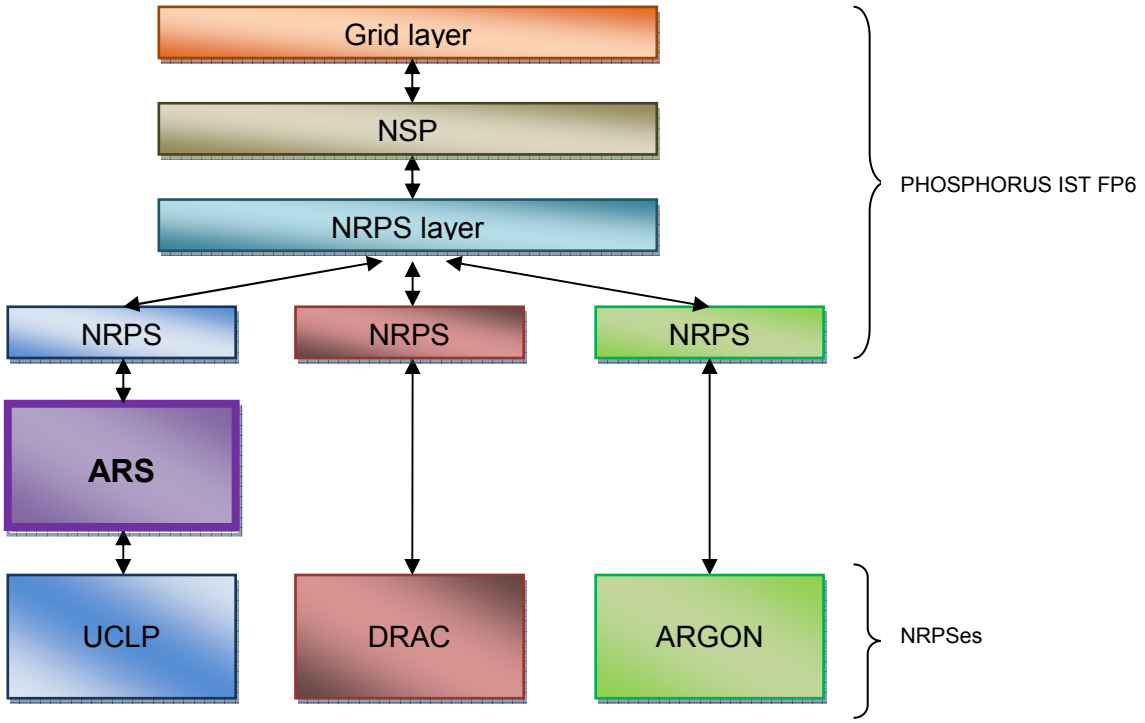


Figure 6. Integration of the UCLP system into the PHOSPHORUS IST FP6 project by means of the ARS

The figure number 8 shows the architecture of the system described in this documentation. Two types of users can interact with the ARS (it will be explained more deeply in the next section) to make the queries to the system. The ARS contacts the UCLP system to get the topology of the network through the 'topology file' when the service is started up. When a connection has to be created or deleted because a user has requested it, the ARS contacts the UCLP system to delete or create the corresponding lightpaths (connections) through the LP-WS (LightPath Web Service).

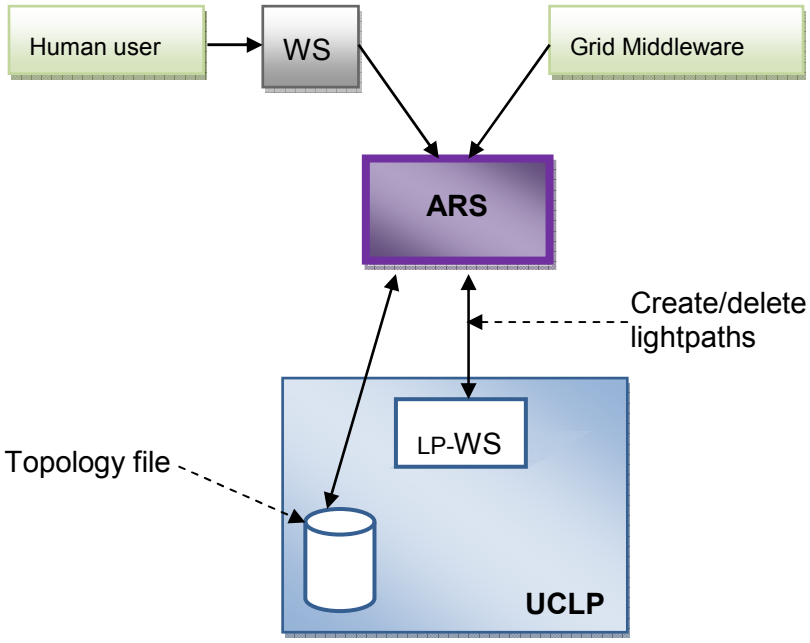


Figure 7. Architecture of the System

2.3. MOTIVATION

Advance reservations are a useful network service that can not only provide guaranteed connectivity for network users, but also allow service providers to more efficiently manage network resources and thus, provide a better quality of service.

In general, two types of resource reservations in networks can be distinguished: immediate reservations, which are made in a just-in-time manner, and advance reservations, which allow reserving resources a period of time before they are used. Actually, immediate reservations are a special case of advance reservations, where the start time requested is the "present time". Advance reservations are useful in any application where large amounts of data have to be transmitted over a network and the time of such a transmission is known in advance. This is the case e.g. in web caching or distributed multimedia applications [9] where large amounts of content such as video files have to be transmitted up to a certain, predefined deadline. Another example is grid computing, where typical computations on the distributed parallel systems result in large amounts of data which also have to be transmitted in time between different machines. In order to allocate the resources that will create the optimal path through the network, routing strategies are based on Dijkstra's Shortest Path (DSP) algorithm.

These reasons explain why the PHOSPHORUS IST FP6 project offers the possibility to make advance reservations. But to offer this service to all the users of the different systems integrated, these ones must support advance reservations, and as UCLP had to be integrated in the project and it didn't supports the advance reservations, it was necessary to implement a web service with this functionality.

2.4. GOALS

In this section the goals of the project described in this documentation and the personal ones are exposed.

2.4.1. Goals of the project

The main goal of the project is to provide the UCLP system with the capability to make advance reservations of the network resources. In this way, UCLP will permit its users to make advance reservations and will be integrated in the PHOSPHORUS IST FP6 project.

Other goals of functionality are the following ones:

- an efficient web service
- a robust web service

2.4.2. Personal goals

There are some personal goals that I have tried to achieve:

- Use the knowledge acquired during the degree.
- Learn more about the network technologies.
- Learn more about the use of web services.
- Finish the degree.
- Go on working in the i2CAT Foundation.

3. Analysis of Predecessors and Feasibility

3.1. Analysis of Predecessors

3.1.1. Similar systems

As explained in section 2.1.2, the other systems that the PHOSPHORUS ISTFP6 integrates are ARGON and DRAC. Both systems already support advance reservations. Following there is a little summary of these systems.

ARGON

The German research project VIOLA (Vertically Integrated Optical testbed for Large Applications) aims at the development of new mechanisms for dynamic and advance user bandwidth allocation and reservation in a heterogeneous multi-vendor, multi-layer network infrastructure. In this context recent signalling mechanisms and network services will be evaluated as a basis for the next generation networks. Apart from the test of multi-vendor network equipment and the development of software tools for bandwidth provisioning, the enhancement and test of Grid applications is another major focus of the project.

A network reservation system needs to reconcile the requirements of the Grid applications with the traffic engineering capabilities and services of the network and its containing entities. The reservation system has to introduce an advance reservation mechanism as network devices lack a concept of time. This mechanism should be applicable for single domain reservations as well as for network reservations involving multiple domains. Inter-domain reservations can be used with several signalling options and path computation techniques. This network reservation system is also referred to as ARGON: Allocation and Reservation in Grid-enabled Optic Networks.

The main objective of the network reservation system ARGON is to provide a service for scientific computing. The service can be used by network-aware distributed applications that have specific demands on the network. In this scenario the network is used to connect sites, nodes, or particular networks in a point-to-point or point-to-multipoint manner with a certain level of QoS.

In order to fit the needs of coordinated computation in an interconnected world, two types of reservations are identified: immediate (or on-demand) reservations and in advance reservations. The latter is motivated by the fact that computing resources are booked in advance. Additionally, workflows have to be regarded when resources are scheduled. As tasks in the scope of scientific computing often have dependencies such that tasks are performed concurrently or sequentially, this has to be reflected by a network reservation system, i.e. a workflow is realized by a set of reservations that may depend on each other. Reservations describe a certain QoS that has to be mapped on services provides by the network or single entities. Beside categories concerning bandwidth and delay issues, aspects like levels of protection may be present.

DRAC

This project has been embarked by Nortel. The primary goal of the DRAC project is to enable a high degree of coupling between applications and networks, resulting in an improved application network experience, while optimizing equipment investments and operational expenses. These new capabilities provide applications the means to directly drive their share of network resources within a policy defined envelope of

flexibility. Network resources include bandwidth, QoS, security, acceleration appliances, sensors, and more.

DRAC makes a strong departure from point-and-click human-driven interfaces and network operators. Likewise, DRAC moves beyond the capex and opex inefficiencies seen in statically and peak-provisioned network setups. DRAC provides a service interface with an abstracted view of the network. It enables applications to control the network, yet without requiring applications to interface directly with a wide range of diverse and constantly evolving network protocols, features, and devices. The interface to applications is bi-directional, enabling network performance and availability information to be abstracted upwards towards the application. DRAC's strategy is to use existing standards and toolsets for these interfaces to greatly simplify deployment in multi-vendor and multi-technology environments.

DRAC is implemented as middleware layered between the application and the network elements. It is designed to be portable to any Java platform. The DRAC platform is highly extensible, both in its northbound services rendered to applications, and in its southbound interfaces to network elements be it Optical, Wireline, Wireless, etc. The DRAC core framework includes a policy engine, AAA services, a QoS manager, a resource monitor, and a topology discovery engine.

3.1.2. Algorithms used

DSP

One of the wishes of this project is to calculate the optimal path for the connections requested trough the network in an efficient way. The most known algorithm to perform this task is the Dijkstra's Shortest Path (DSP) algorithm. So, to calculate the optimal path through the network, routing strategies based on DSP algorithm have been used. To implement the algorithm that calculates the optimal path in the conditions requested in this project, I have followed the indications given in the following papers:

- Lars-Olof Burchard (2003, February). "Source Routing Algorithms for Networks with Advance Reservations". Communication and Operating Systems, Technische Universitaet Berlin. ISSN 1436-9915.
- Jun Zheng, University of Ottawa. "Toward Automated Provisioning of Advance Reservation Service in Next-Generation Optical Internet". Baoxian Zhang, Graduate University of the Chinese Academy of Sciences Hussein T. Mouftah, University of Ottawa.
- Chin Guok, David Robertson, Mary Thompson, Jason Lee, Brian Tierney and William Johnston. "Intra and Interdomain Circuit Provisioning Using the OSCARS Reservation System". Energy Sciences Network. Berkeley, California 94720.
- Sumit Naiksatam, Silvia Figueira, Stephen A. Chiappari and Nirdosh Bhatnagar. "Modeling and Simulation of Advance Reservations in Dynamically Provisioned Optical Networks". Department of Computer Engineering, Department of Applied Mathematics. Santa Clara University.
- Jun Zheng and H. T. Mouftah, Fellow, IEEE. "Supporting Advance Reservations in Wavelength-Routed WDM Networks". Department of Electrical and Computer Engineering. Queen's University. Kingston, ON K7L 3N6, Canada.
- Silvia Figueira, Neena Kaushik, Sumit Naiksatam,Stephen A. Chiappari, and Nirdosh Bhatnagar. "Advance Reservation of Lightpaths in Optical-Network Based Grids". Department of Computer Engineering, Department of Applied Mathematics. Santa Clara University.

To understand how the algorithm is used in this project works, firstly I'm going to sum up how the traditional DSP works.

Dijkstra's algorithm, when applied to a graph, quickly finds the shortest path from a chosen source to a given destination. In fact, the algorithm is so powerful that it finds all shortest paths from the source to all destinations. This is known as the *single-source* shortest paths problem. In the process of finding all shortest paths to all destinations, Dijkstra's algorithm will also compute a *spanning tree* for the graph. While an interesting result in itself, the spanning tree for a graph can be found using lighter (more efficient) methods than Dijkstra's.

How It Works

First the entities used are going to be defined. In this context, the graph is made of *nodes* and *edges (links)* which link nodes together. Links are directed and have an associated *distance*, sometimes called the weight or the cost. The distance between the vertex *u* and the vertex *v* is noted [*u*, *v*] and is always positive.

Dijkstra's algorithm partitions vertices in two distinct sets, the set of *unsettled* vertices and the set of *settled* vertices. Initially all vertices are unsettled, and the algorithm ends once all vertices are in the settled set. A vertex is considered settled, and moved from the unsettled set to the settled set, once its shortest distance from the source has been found.

The following data structures are used for this algorithm:

d stores the best estimate of the shortest distance from the source to each vertex

π stores the predecessor of each vertex on the shortest path from the source

S the set of settled vertices, the vertices whose shortest distances from the source have been found

Q the set of unsettled vertices

With those definitions in place, a high-level description of the algorithm is deceptively simple. With *s* as the source vertex:

```
// initialize d to infinity, π and Q to empty
d = ( ∞ )
π = ()
S = Q = ()

add s to Q
d(s) = 0

while Q is not empty
{
     u = extract-minimum(Q)
     add u to S
     relax-neighbors(u)
}

relax-neighbors(u)
{
     for each vertex v adjacent to u, v not in S
     {
          if d(v) > d(u) + [u,v]    // a shorter distance exists
          {
               d(v) = d(u) + [u,v]
               π(v) = u
```

```
                add v to Q
            }
        }
}

extract-minimum(Q)
{
    find the smallest (as defined by d) vertex in Q
    remove it from Q and return it
}
```

The cost of the algorithm is $\Theta(n^3)$, where *n* is the number of the vertexes of the path, because calculates the shortest path for each vertex.

DSP in the ARS

To adapt the traditional DSP to the ARS, a mix of the algorithms proposed in the papers mentioned before has been performed. Thus, in the ARS the algorithm works as follows.

To calculate the optimal path in this project, before computing the DSP algorithm, a selection of the available links is done. In other words, from all the links of the network, only the ones that are available for the time requested are selected. So, the network in which the DSP algorithm will be computed is the original one without the links not selected. In the Design section (number 8) the link selection process will be explained in detail.

3.2. FEASIBILITY

The service described in this documentation allows the users of the UCLP system to make advance reservations of the network resources. Therefore, the service allows the system to manage, in a more flexible and intelligent way, scarce resources such as the bandwidth of a network, and this helps to reduce the exploitation and working costs of the network. Moreover, advance reservations are a useful network service that provide guaranteed connectivity for network users and allow service providers to more efficiently manage network resources and thus, provide a better quality of service.

Another important fact that should be emphasized is that computing resources are usually booked in advance. Additionally, workflows have to be regarded when resources are scheduled. As tasks in the scope of scientific computing often have dependencies such that tasks are performed concurrently or sequentially, this has to be reflected by a network reservation system. So, it is important to offer this type of service.

4. REQUIREMENTS ANALYSIS

Requirements Analysis encompasses those tasks that go into determining the needs or conditions to meet for a new or altered device, taking account of the possibly conflicting requirements of the various stakeholders, such as beneficiaries or users. Requirements analysis is critical to the success of a development project. Systematic requirements analysis is also known as *Requirements Engineering.*

The requirements are a description of the wishes and needs of the product. The main goal of the requirements analysis is to identify and documenting what is what we really need to communicate to the client and with the others members of the development team. It is very important to define the requirements without confusions in order to recognize all the risks. The requirements must be actionable, measurable, testable, related to identified business needs or opportunities, and defined to a level of detail sufficient for system design.

Requirements engineering, also can be defined as the systematic process of:

- analysing the problem
- documenting the resulting observations in a variety of representation formats, and
- checking the accuracy of the understanding gained.

The hardest single part of building a software system is deciding precisely what to build. The methodology used to specify the requirements is the Volere Methodology [20].

The goal is to define the basic interaction between the users and the blocks that the system builds. At this point, no reference to the architecture or specific blocks will be done yet, since this section is focused on clarifying the basic interactions between the software/human users and the system to be developed.

4.1. THE CLIENT, THE COSTUMER AND OTHER STAKEHOLDERS

4.1.1. The client

The client is the person who makes the investment in this product. So, in this case, the client is the i2CAT Foundation.

4.1.2. The costumer

The costumer is the person intended to buy the product. In this project the costumer is formed by the PHOSPHORUS Consortium.

4.1.3. Other stakeholders

The stakeholders are the roles of other people and organizations who are affected by the product, or whose input is needed to build the product. In this case, the groups involved in this project are the following ones: Communication Research Center (CRC, Canada), Inocybe Technologies Inc. (Canada), University of Bonn (Germany), Surfnet (The Netherlands), FHG (Germany), and University of Amsterdam (The Netherlands).

4.2. USERS OF THE PRODUCT

The users of this product can be classified in two groups: the ones that use the product as soon as it will be finished, and the ones that maybe will use it in the future:

- Immediate Users: are basically the ones that are in the PHOSPHORUS Consortium, which can be either a human user or Grid Middleware (MetaScheduler).
- In the future, possible users could be Network Operation Centers (NOC) and E-science users.

In both cases, any type of user is able to perform all the functionalities and both have the same privileges. Both actors are also the initiators of the use cases.

Characteristics of the users:

- *Number*: limited by the number of the users of the net.
- *Experience*: no experience needed.
- *Localization*: They will access the system from the server required.

Another actor is a scheduler (a clock) that informs the system when a connection must be activated or deleted. The scheduler is programmed by the service and is a part of the system. It will be the initiator of some use cases that must be called automatically, without any event from the other actors (see section 4.4).

- Number: one scheduler is programmed at once.
- Experience: -
- Localization: it is in the service.

4.3. NAMING CONVENTIONS AND DEFINITIONS

One of the main requirements of the project is reservation management. The user has to be able to create, delete and query advance reservations in the network through the system.

An advance reservation is composed by a set of services and the services are composed by one or more connections. To define these items the following information is needed:

- Reservation: A reservation is defined by the following parameters:
  - Reservation ID: unique identifier of the reservation.
  - isPre: indicates if the reservation is a pre-reservation (true) or a permanent one (false).
  - Notification consumer URL: URL of a Notification Consumer that is to be notified when any of the services' status changes.
  - Owner: the owner of the reservation.
  - Services: the set of services.
- Job: Several independent reservations can be grouped to a single "job". This feature basically allows for a higher convenience when complex workflows are planned: Planning of workflow items depends on how exactly the constraints of preceding workflow items are set such that the required resources are available. While the workflow is not yet completely planned, these workflow items are only loosely blocked as "pre-reservations", their affiliation to the same job is indicated by a common job ID. Therefore, if the complete workflow was

successfully planned, these pre-reservations can be modified to regular reservations. In the same manner, the complete set of pre-reservations can be cancelled with a single request in case the workflow cannot be implemented. These functionalities will be explained better in the section 4.4. The following parameters define a job:

  - Job ID: unique identifier of a job.
  - Pre-reservations: a set of pre-reservations grouped into the job.

- Connection: A connection is defined by the following parameters:
  - Connection ID: This is a unique identifier for a connection within a set of connections aggregated to a service.
  - Two or more endpoints: Exactly one of these endpoints is tagged as source endpoint, and other is tagged as destination (or target) endpoint. This tag is important for unidirectional and bidirectional trees.
  - Bandwidth/delay constraints: Minimum and maximum bandwidth (which may be equal), maximum latency and the actual bandwidth.
  - Directionality (in the case of exactly two endpoints, the unidirectional tree is simply a unidirectional connection and both the bidirectional tree and the full mesh can be reduced to a bidirectional connection):
    - Unidirectional tree: There are unidirectional connections from the “source endpoint” to each of the “target endpoints”. Each of these connections fulfils the given bandwidth/delay constraints.
    - Bidirectional tree: The “source endpoint” has a bidirectional connection fulfilling the bandwidth/delay constraints in each direction to every “target endpoint”.
    - Full mesh: Every endpoint is connected to every other endpoint with the given bandwidth/delay constraints.
  - Status: the status of the connection, which may be unknown, pending, active, completed, cancelled by user, cancelled by system, setup in progress and teardown in progress
  - EPR Wrapper: the endpoint reference, necessary to create the connection physically when the service contacts to the UCLP system.
- Service: While “connections” are of topological nature, “services” add time constraints. A service is defined by a service type and consists of one or more connections. All connections grouped in a service are characterised by exactly the same time constraints.
  - Service ID: This is a unique identifier for a service within a set of services aggregated to a single reservation.
  - Automatic activation: indicates if the connections of the service have to be activated automatically when its start time arrives or not.
  - Type of Reservation: Reservations can be fixed, deferrable or malleable. A fixed reservation has a fixed starting time and a fixed duration. A deferrable reservation also has a fixed duration, but a variable starting time. A malleable reservation finally has a variable starting time, a variable duration and, as a result, variable bandwidth. These three reservation types are visualized in the following figure.

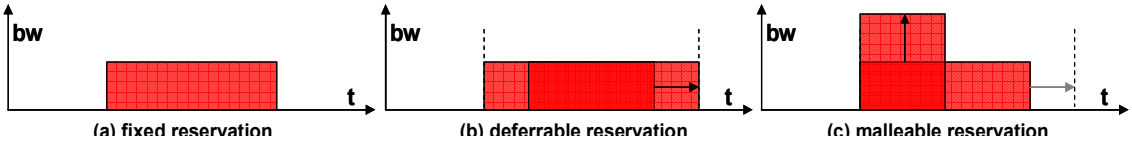


Figure 8. Types of reservations

- Time constraints (note that bandwidth/delay constraints are given in the connection data).
  - For fixed reservations: Start time and duration.
  - For deferrable reservations: Earliest start time, duration, and deadline.
  - For malleable reservations: Earliest start time, amount of data to be transferred, and deadline (time by which data completely has to be transferred). Higher layers should not expect that a fixed bandwidth is reserved for the whole duration of the connection. Instead, the bandwidth assignment can be variable over time.
- Connections: the set of connections.

The system must store, retrieve, modify and delete the resource-related information according to the topology of the network reservations. This will require the implementation of a persistence module and a data model to maintain the information about the topology and the reservations. The definition of Node, Link and Endpoint that is going to be used in this documentation is the following one:

- Node: a node is a physical point in the network that contains different endpoints. The information required for a node should include:
  - Node ID: identifier of the node, that is unique in the network
  - Name: provided by the user that creates the node object.
- Endpoint: an endpoint is an edge port of a node; it can be a port connected to another node (internal endpoint) or a port presented to the client of the network (external endpoint). The endpoints that interconnect two nodes will form part of a link. Otherwise, the other ports will be selectable to be the source or destination ports of a reservation. The information required for an endpoint should include:
  - Endpoint ID: a unique endpoint identifier
  - IP address: the IP address of the port
  - Name of the endpoint.
  - Description of the endpoint.
  - IntrefaceType: the type of the endpoint (which can be border, an endpoint connected to another network domain, or user, an endpoint connected to a user site)
  - Bandwidth: the bandwidth of the port in Mbps
  - Is External: indicates if the endpoint is external (true) or not (false). The external endpoints are the border and user ones. The endpoints considered as internal endpoints are the ones that form part of a link.
  - EPR Wrapper: the endpoint reference, used to identify an endpoint of the ARS with an endpoint of the UCLP system

- Link: a link represents a physical connection between two different nodes. It is equivalent to the lightpaths in the UCLP system. It is associated to the two endpoints connected physically. The information required for a link should include:
    - Link ID: a unique identifier
    - Name: the name provided by the user that creates the link object.
    - Endpoints: the source and destination endpoints of the link.
    - Bandwidth: the bandwidth of the link
    - Delay: the delay of the link (this parameter and the bandwidth will be used to obtain the weight required when a route has to be calculated using the Dijkstra's shortest path algorithm).

4.3.1. Acronyms

AAA	Authentication, Authorization, Accounting
API	Application Programming Interface
APN	Articulated Private Network
ARGON	Allocation and Reservation in Grid-enabled Optic Networks
ARS	Advance Reservation Service
ASON	Automatically Switching Optical Network
BGP	Border Gateway Protocol
BPEL	Business Process Execution Language
BSD	Berkeley Software Distribution
BW	bandwidth
Capex	Capital expenditures
CBDF	Cross Border Dark Fiber
CRC	Communications Research Centre, Canada
DB	database
DRAC	Dynamic Resource Allocation Controller
DSP	Dijkstra's Shortest Path
DWDM	Dense Wavelength Division Multiplexing
EPR	EndPoint Reference
FHG	Fraunhofer-Gesellschaft
FP6	Framework Program 6th
G2MPLS	Grid-enabled GMPLS
GLIF	Global Lambda Integrated Facility
GMPLS	Generalized Multi Protocol Label Switching
GMPLS CP	GMPLS based Control Plane
GNS	Grid Network Services
GUI	Guided User Interface

HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
ID	Identifier
IST	Information Society Technology
JS	Java Script
JSP	Java Server Page
LP-WS	LightPath Web Service
MPLS	Multi Protocol Label Switching
NREN	National Research and Education Networks
NRPS	Network Resources Provisioning Systems
NSP	Network Service Plane
ONDM	Optical Networking Design and Modelling
ONS	Optical Network Services
Opex	operational expenditure
ORDBMS	object-relational database management system
ORM	object-relational mapping
OS	Operating System
PSNC	Poznan Supercomputing and Networking Center
QoS	Quality of Service
RFP	Request For Proposals
RUP	Rational Unified Process
SC	SuperComputing
SQL	Structured Query Language
TN	Transport Network
UCLP	User Controlled Lightpath Provisioning
UNICORE	Uniform Interface to Computing Resources
UofO	University of Ottawa, Canada
UP	Unified Process
URL	Uniform Resource Locator
VIOLA	Vertically Integrated Optical testbed for Large Applications
VLAN	Virtual Local Area Network
WP	Work Package
WRSF	Wire Rope Safety Fence
WS	Web Service
XML	eXtensible Markup Language

4.4. The Scope of the Product

In this subsection the use cases of the product are identified and defined.

4.4.1. Use Case Diagram

A use case diagram is a type of behavioural diagram. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases) and any dependencies between those use cases.

In this subsection the primary actions that the users will execute in the system are going to be described. These actions are the initiators of the processes that will be performed in the system in order to offer the required functionalities.

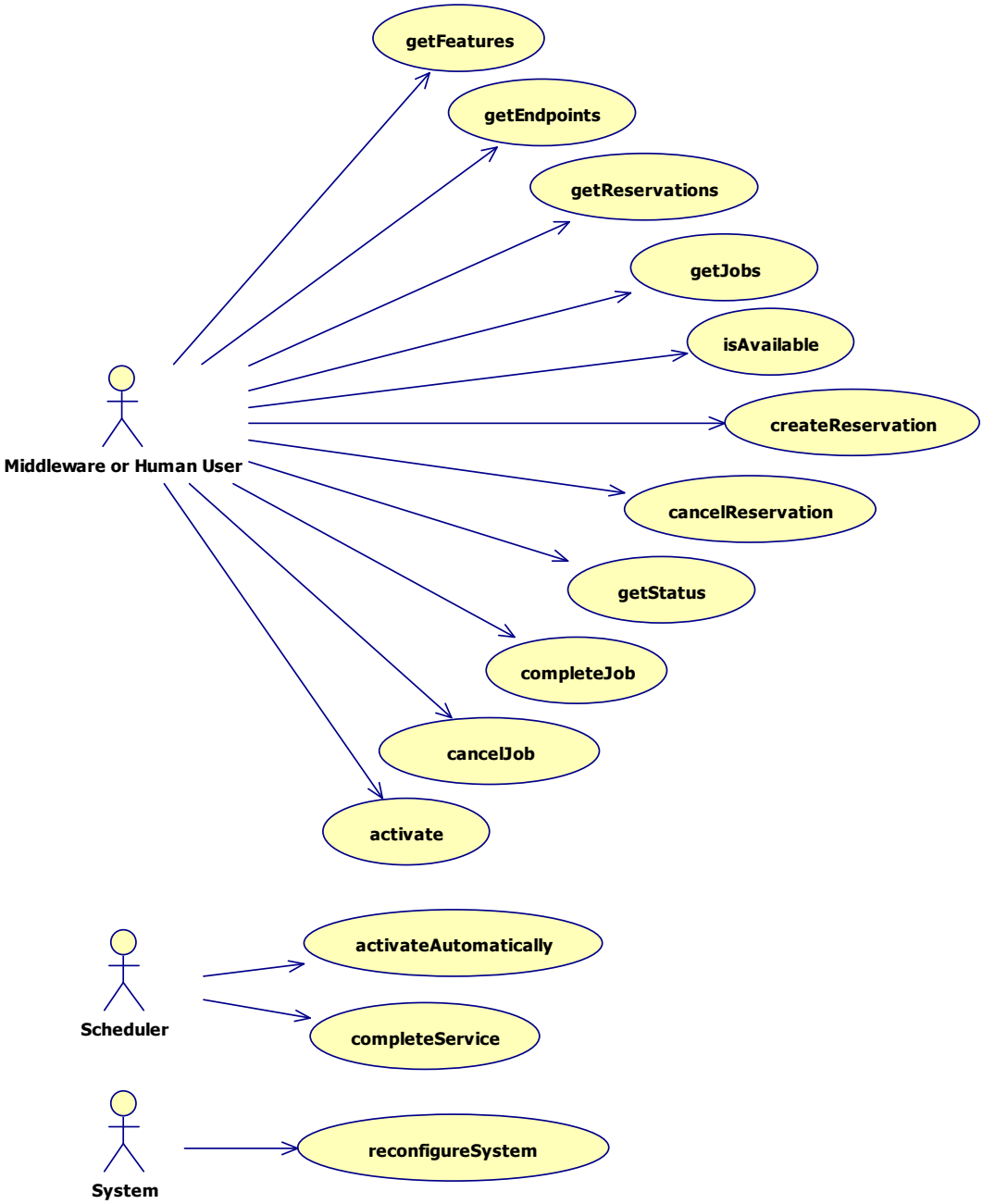


Figure 9. Use case diagram

4.4.2. Use Case Specification

A use case is a document that describes a sequence of events that an actor that uses the system performs in order to begin a process that he wants to start. In this section the actions of each use case are described briefly.

I. Get Features

Use Case: GetFeatures

Actors: Human User or Grid Middleware (initiator)

Goal: Retrieve information about the features of the system.

Abstract: The user requests the features of the system.

Type: primary and essential.

Basic flow:

Actions of the actors	Response of the system
1. The user requests the features available in the system.	
	2. The system authenticates the user.
	3. The system retrieves the information about the features of the system and returns this information to the user.

Alternative flow:

Line 2. If the user has no privilege to make the request, an error is returned.

Crossed references:

II. Get Endpoints

Use Case: GetEndpoints

Actors: Human User or Grid Middleware (initiator)

Goal: Retrieve information about the endpoints of the network.

Abstract: The user requests the endpoints of the network.

Type: primary and essential.

Basic flow:

Actions of the actors	Response of the system
1. The user requests the endpoints to know the availability before making a reservation.	
	2. The system authenticates the user.

	3. The system retrieves the endpoints from the database and returns their information to the user.

Alternative flow:

Line 2. If the user has no privilege to make the request, an error is returned.

Crossed references:

III. Get Reservations

Use Case: GetReservations

Actors: Human User or Grid Middleware (initiator)

Goal: Retrieves information about the reservations done in the system.

Abstract: The user requests the reservations done.

Type: primary and essential.

Basic flow:

Actions of the actors	**Response of the system**
1. The user requests the reservations done in order to know the resources requested until the moment, and the status of each reservation done.	
	2. The system authenticates the user.
	3. The system retrieves the information about the reservations done and returns their information to the user.

Alternative flow:

Line 2. If the user has no privilege to make the request, an error is returned.

Line 3. If there are no reservations, the system returns a message indicating it.

Crossed references:

IV. Get Jobs

Use Case: GetJobs

Actors: Human User or Grid Middleware (initiator)

Goal: Retrieves information about the jobs.

Abstract: The user requests the jobs existent in the database.

Type: primary and essential.

Basic flow:

Actions of the actors	Response of the system
1. The user requests the jobs in order to know the pre-reservations done.	
	2. The system authenticates the user.
	3. The system retrieves the information about the jobs and its pre-reservations done and returns their information to the user.

Alternative flow:

Line 2. If the user has no privilege to make the request, an error is returned.

Line 3. If there are no jobs, the system returns a message indicating it.

Crossed references:

V. Is Available

Use Case: isAvailable

Actors: Human User or Grid Middleware (initiator)

Goal: Checks for the availability of network resources.

Abstract: The user requests to the system if some network resources are available at the time request.

Type: primary and essential.

Basic flow:

Actions of the actors	Response of the system
1. The user indicates all the parameters for the reservation he wants to check.	
	2. The system authenticates the user.
	3. The system checks in the database if the resources are available.
	4. If the entire process has been successful, the system returns an OK message

Alternative flow:

Line 2. If the user has no privilege to make the request, an error is returned.

Line 3. If there are not enough resources, the system returns a message indicating it.

Crossed references:

VI. Create Reservation

Use Case: createReservation

Actors: Human User or Grid Middleware (initiator)

Goal: Make a reservation of network resources.

Abstract: The user requests to the system to reserve some network resources at the time request.

Type: primary and essential.

Basic flow:

Actions of the actors	Response of the system
1. The user indicates the resources that have to participate in the reservation (maybe after a previous "Is Available" request) and gives the time-related restrictions for the reservation.	
	2. The system authenticates the user.
	3. The system checks in the database if the resources are available.
	4. If there are no errors, the system schedules the reservation and registers the related information in the database.

Alternative flow:

Line 2. If the user has no privilege to make the request, an error is returned.

Line 3. If resources are not available, the system returns an error.

Crossed references:

VII. Cancel Reservation

Use Case: cancelReservation

Actors: Human User or Grid Middleware (initiator)

Goal: Cancels a reservation.

Abstract: The user requests to the system to cancel a reservation. Depending on the status of the reservation the system will call the UCLP system to delete the connections or not.

Type: primary and essential.

Basic flow:

Actions of the actors	Response of the system
1. The user indicates the reservation to be cancelled.	

	2. The system authenticates the user. 3. The system checks that the reservation exists. 4. The system contacts the UCLP system and calls the appropriate cancellation function. 5. If there are no errors from the UCLP system, the system deletes the related information of the database.

Alternative flow:

Line 2. If the user has no privilege to make the request, an error is returned.

Line 3. If the indicated reservation doesn't exist, an error is returned.

Crossed references:

VIII. Get Status

Use Case: GetStatus

Actors: Human User or Grid Middleware (initiator)

Goal: Gets the status of a reservation.

Abstract: The user requests the status of a reservation to the system.

Type: primary and essential.

Basic flow:

Actions of the actors	**Response of the system**
1. The user indicates the reservation that he wants to query.	
	2. The system authenticates the user.
	3. The system checks that the reservation exists.
	4. The system retrieves the information from the database and presents it to the user.

Alternative flow:

Line 2. If the user has no privilege to make the request, an error is returned.

Line 3. If the indicated reservation doesn't exist, an error is returned.

Crossed references:

IX. Complete Job

Use Case: CompleteJob

Actors: Human User or Grid Middleware (initiator)

Goal: Completes a job.

Abstract: The user wants to complete a job and indicates the identifier of the job. The system converts all the pre-reservations of the job in permanent reservations.

Type: primary and essential.

Basic flow:

Actions of the actors	Response of the system
1. The user indicates the JobID of the job to be completed.	
	2. The system authenticates the user.
	3. The system checks if the job exists.
	4. The system converts all pre-reservations made in the context of this job to permanent reservations.
	5. The system schedules the reservations and registers the related information in the database.
	6. The system deletes the job from the DB.

Alternative flow:

Line 2. If the user has no privilege to make the request, an error is returned.

Line 3. If the job does not exist, the system returns an error message.

Line 5. If the UCLP system returns an error, the reservations are not scheduled and an error is returned.

Crossed references:

X. Cancel Job

Use Case: CancelJob

Actors: Human User or Grid Middleware (initiator)

Goal: Cancels a job.

Abstract: The user wants to cancel a job and indicates the identifier of the job.

Type: primary and essential.

Basic flow:

Actions of the actors	Response of the system
1. The user indicates the JobID of the job to be cancelled.	
	2. The system authenticates the user.
	3. The system checks if the job exists.
	4. The system cancels all pre-reservations made in the context of the given job and deletes the job from the DB.

Alternative flow:

Line 2. If the user has no privilege to make the request, an error is returned.

Line 3. If the job does not exist, the system returns an error message.

Crossed references:

XI. Activate

Use Case: Activate

Actors: Human User or Grid Middleware (initiator)

Goal: Activates a service of a reservation.

Abstract: The user indicates the service he wants to activate. The system creates all the connections of this service.

Type: primary and essential.

Basic flow:

Actions of the actors	Response of the system
1. The user indicates the reservationID of the reservation and the serviceID of the service to activate.	
	2. The system authenticates the user.
	3. The system checks if the reservation and the service exist.
	4. The system sends the activation message to the UCLP system.
	5. The UCLP system establishes the required connections for that service and the system updates the database with the required information.

Alternative flow:

Line 2. If the user is not allowed to activate the reservation, the system returns an error.

Line 3. If the reservation or the service does not exist, the system returns an error.

Line 4. If the UCLP system has some problem to establish the connections, an error is returned and the system rolls-back the activations already done.

Crossed references:

XII. Activate Automatically

Use Case: Activate Automatically

Actors: Scheduler (initiator)

Goal: Activates a service of a reservation.

Abstract: The start time request to activate a service has been arrived. The scheduler indicates it to the system and this one creates all the connections of this service.

Type: secondary and essential.

Basic flow:

Actions of the actors	Response of the system
1. The scheduler calls the system to activate a service.	
	2. The system sends the activation message to the UCLP system.
	3. The UCLP system establishes the required connections for that service and the system updates the database with the required information.

Alternative flow:

Line 3. If the UCLP system has some problem to establish the connections, an error is returned and the system rolls-back the activations already done.

Crossed references:

XIII. Complete Service

Use Case: Complete Service

Actors: Scheduler (initiator)

Goal: Completes a service of a reservation.

Abstract: The end time to complete a service has been arrived. The scheduler indicates it to the system and this one deletess all the connections of this service.

Type: secondary and essential.

Basic flow:

Actions of the actors	Response of the system
1. The scheduler calls the system to delete a service.	
	2. The system sends the completing message to the UCLP system.
	3. The UCLP system deletes the required connections for that service and the system updates the database with the required information.

Alternative flow:

Line 3. If the UCLP system has some problem to delete the connections, an error is returned and the system rolls-back the activations already done.

Crossed references:

XIV. Reconfigure System

Use Case: Reconfigure System

Actors: System

Goal: Reconfigure the system if some concepts have changed.

Abstract: When the service is started, it has to be checked if the net has changed, if there were connections active, if the start time of a service has been passed, etc.

Type: secondary and essential.

Basic flow:

Actions of the actors	Response of the system
1. The system is started up.	
	2. The system checks if the net has been changed.
	3. If the net has been changed, the path of the connections that were “pending” or “active” is recalculated.
	4. If the start time of connections that were “pending” hasn’t arrived, the scheduler is reprogrammed.
	5. If the start time of connections that were “pending” has arrived, but not the end time, the system calls the UCLP system to create them physically.
	6. If the start time of connections that were “pending” has arrived, and also the end time, the connection is

cancelled.

7. If the end time of connections that were “active” has arrived, the system calls the UCLP system to delete them physically.
8. If the end time of connections that were “active” hasn’t arrived, the system calls the UCLP system to create them physically again, and the scheduler is programmed to delete them when its end time arrive.

Alternative flow:

Crossed references:

4.5. SPECIFIC REQUIREMENTS

4.5.1. Functional Requirements

The functional requirements specify what the system must do: the services, the offers and the interaction with the users. They are divided in two categories: evident and hidden.

- *Evident*: the user knows that this requirement is being performed.
- *Hidden*: The user doesn’t know that this requirement is being performed.

Below, the description of the functional requirements can be seen. The template from Volere, applying some changes in order to adapt this template to this thesis, has been used. Following there is a little explanation about what contains each requirement definition:

- Requirement#: is a unique identifier of the requirement.
- Requirement Type: can be hidden or evident.
- Use Case#: is the use case which the requirement refers to. To see which use case is each number, see the section number 4.4.2.
- Description: is a little explanation about the intention of the requirement.
- Rationale: is the justification of the requirement.
- Source: is the user who raised the requirement.
- Fit criterion: is a measurement of the requirement such that it is possible to test it.
- Customer satisfaction: is a degree of stakeholder happiness if this requirement is successfully implemented.
- Customer dissatisfaction: is a measure of stakeholder unhappiness if this requirement is not part of the final product.
- Dependencies: are the other requirements that this one depends on them.

Requirement #: 1 **Requirement Type**: evident **Use Case#**: 1
Description: The user will be able to consult the features of the system.
Rationale: It is necessary to know the available features of the system in order to know which functionalities the user can call.
Source: Human user or Middleware.
Fit Criterion: The user can ask for the features of the system at any time.
Customer Satisfaction: 3 **Customer Dissatisfaction**: 5
Dependencies:

Requirement #: 2 **Requirement Type**: evident **Use Case#**: 2
Description: The user will be able to consult the available endpoints of the network.
Rationale: It is necessary to know the available endpoints of the network in order to create the connections between these endpoints.
Source: Human user or Middleware.
Fit Criterion: The user can ask for the available endpoints at any time.
Customer Satisfaction: 3 **Customer Dissatisfaction**: 5
Dependencies: requirement #1

Requirement #: 3 **Requirement Type**: evident **Use Case#**: 3
Description: The user will be able to consult the reservations done of the network resources.
Rationale: It is interesting to know which reservations have been done in order to be able to cancel these reservations or to query its status.
Source: Human user or Middleware.
Fit Criterion: The user can ask for the reservations done at any time.
Customer Satisfaction: 3 **Customer Dissatisfaction**: 3
Dependencies: requirement #1

Requirement #: 4 **Requirement Type**: evident **Use Case#**: 4
Description: The user will be able to consult the jobs stored in the database.
Rationale: It is interesting to know which jobs have been done in order to be able to cancel these jobs or to complete them.
Source: Human user or Middleware.
Fit Criterion: The user can ask for the jobs done at any time.
Customer Satisfaction: 3 **Customer Dissatisfaction**: 3
Dependencies: requirement #1

Requirement #: 5 **Requirement Type**: evident **Use Case#**: 5
Description: The user will able to ask for the availability of the network resources at a requested time.
Rationale: It is interesting to know if a reservation is possible before querying to create it.
Source: Human user or Middleware.
Fit Criterion: The user can ask for the availability of a reservation at any time.
Customer Satisfaction: 4 **Customer Dissatisfaction**: 3
Dependencies: requirement #1, #2

Requirement #: 6 **Requirement Type**: evident **Use Case#**: 5
Description: When the user makes an availability of the network resources request, if these resources are not available, an alternative start time must be returned.
Rationale: It is interesting to know if an alternative start time exists for the resources requested.
Source: Human user or Middleware.
Fit Criterion: The alternative start time will be returned every time that an isAvailable request done, is not available for the resources requested.
Customer Satisfaction: 2 **Customer Dissatisfaction**: 2
Dependencies: requirement #5

Requirement #: 7 **Requirement Type**: evident **Use Case#**: 6
Description: The user will be able to create a reservation of the network resources at a requested time if these resources are available.
Rationale: It is necessary to create a reservation of the network resources in order to activate the required connections at the requested time. This requirement is the main requirement of the system.
Source: Human user or Middleware.
Fit Criterion: The user can ask for the creation of a reservation at any time.
Customer Satisfaction: 5 **Customer Dissatisfaction**: 5
Dependencies: requirement #1, #2, #5

Requirement #: 8 **Requirement Type**: evident **Use Case#**: 6
Description: The user will be able to create a pre-reservation of the network resources at a requested time if these resources are available.
Rationale: It is interesting to create pre-reservations of the network resources if the user is not sure that he will use them. These pre-reservations will be grouped in jobs and the user will be able to convert these pre-reservations into permanent reservations.
Source: Human user or Middleware.
Fit Criterion: The user can ask for the creation of a pre-reservation at any time.
Customer Satisfaction: 4 **Customer Dissatisfaction**: 5
Dependencies: requirement #1, #2, #5

Requirement #: 9 **Requirement Type**: hidden **Use Case#**: 6
Description: If a create reservation request is not possible, an exception is thrown.
Rationale: If the resources requested are not available or the request contains some error (like invalid endpoints, invalid start time, and invalid duration), the system will throw an exception.
Fit Criterion: An exception will be thrown every time that a createReservation request is not possible.
Customer Satisfaction: 2 **Customer Dissatisfaction**: 2
Dependencies: requirement #5

Requirement #: 10 **Requirement Type**: evident **Use Case#**: 6
Description: Different pre-reservations can be grouped into a job.
Rationale: When a user wants to create a pre-reservation, can indicate an existence job to associate the pre-reservation with this job or if no job is given, a new job will be created.
Fit Criterion: Any time the user wants to create a pre-reservation, he could choose to associate this reservation to an existent job or not.
Customer Satisfaction: 3 **Customer Dissatisfaction**: 3
Dependencies: requirement #8

Requirement #: 11 **Requirement Type**: hidden **Use Case#**: 12
Description: A service of a reservation will be activated when its start time arrives.
Rationale: When the start time of a service of a reservation arrives, the connections must be activated physically.
Fit Criterion: The connections must be activated physically as soon as possible.
Customer Satisfaction: 3 **Customer Dissatisfaction**: 5
Dependencies: requirement #7

Requirement #: 12 **Requirement Type**: hidden **Use Case#**: 13
Description: A service of a reservation will be completed when its end time arrives.
Rationale: When the end time of a service of a reservation arrives, the connections must be deleted physically.
Fit Criterion: The connections must be deleted physically as soon as possible.
Customer Satisfaction: 3 **Customer Dissatisfaction**: 5
Dependencies: requirement #7

Requirement #: 13 **Requirement Type**: hidden **Use Case#**: 6
Description: When the user requests to create a reservation, if the start time requested is earlier that the actual date, the reservation won't be created and an exception will be thrown.
Rationale: If some parameter of the request is not correct, the reservation cannot be created.
Fit Criterion: -
Customer Satisfaction: 3 **Customer Dissatisfaction**: 3
Dependencies: requirement #7

Requirement #: 14 **Requirement Type**: hidden **Use Case#**: 6
Description: When the user requests to create a reservation, if the duration requested is smaller than zero, the reservation won't be created and an exception will be thrown.
Rationale: If some parameter of the request is not correct, the reservation cannot be created.
Fit Criterion: -
Customer Satisfaction: 3 **Customer Dissatisfaction**: 3
Dependencies: requirement #7

Requirement #: 15 **Requirement Type**: hidden **Use Case#**: 6
Description: When the user requests to create a reservation, if some identifier of the connections of a service is repeated, the reservation won't be created and an exception will be thrown.
Rationale: If some parameter of the request is not correct, the reservation cannot be created.
Fit Criterion: -
Customer Satisfaction: 3 **Customer Dissatisfaction**: 3
Dependencies: requirement #7

Requirement #: 16 **Requirement Type**: hidden **Use Case#**: 6
Description: When the user requests to create a reservation, if some identifier of the services of a reservation is repeated, the reservation won't be created and an exception will be thrown.
Rationale: If some parameter of the request is not correct, the reservation cannot be created.
Fit Criterion: -
Customer Satisfaction: 3 **Customer Dissatisfaction**: 3
Dependencies: requirement #7

Requirement #: 17 **Requirement Type**: hidden **Use Case#**: 6
Description: When the user requests to create a reservation, the reservation created can be one of these types: fixed, deferrable, malleable; otherwise an exception will be thrown.
Rationale: If some parameter of the request is not correct, the reservation cannot be created.
Fit Criterion: -
Customer Satisfaction: 3 **Customer Dissatisfaction**: 3
Dependencies: requirement #7

Requirement #: 18 **Requirement Type**: hidden **Use Case#**: 6
Description: When the user requests to create a reservation, the directionality of a connection can be 0(unidirectional), 1 (bidirectional) or 3 (full mesh); otherwise an exception will be thrown.
Rationale: If some parameter of the request is not correct, the reservation cannot be created.
Fit Criterion: -
Customer Satisfaction: 3 **Customer Dissatisfaction**: 3
Dependencies: requirement #7

Requirement #: 19 **Requirement Type**: hidden **Use Case#**: 6
Description: When the user requests to create a reservation, the source and destination endpoints of each connection must be valid; otherwise an exception will be thrown.
Rationale: If some parameter of the request is not correct, the reservation cannot be created.
Fit Criterion: -
Customer Satisfaction: 3 **Customer Dissatisfaction**: 3
Dependencies: requirement #7

Requirement #: 20 **Requirement Type**: evident **Use Case#**: 6
Description: When the user requests to create a reservation, the 'automatic activation' can be set as true or false.
Rationale: The user is able to request an automatic activation for a service when its start time arrives, or to request a manual activation.
Fit Criterion: -
Customer Satisfaction: 3 **Customer Dissatisfaction**: 3
Dependencies: requirement #7

Requirement #: 21 **Requirement Type**: hidden **Use Case#**: 6
Description: When a reservation is created, until this reservation is activated, its status is 'pending'.
Rationale: When a reservation is activated, because its start time has arrives or because the user has request the activation manually, the status is set as 'active', but before this activation is set as 'pending'.
Fit Criterion: -
Customer Satisfaction: 3 **Customer Dissatisfaction**: 3
Dependencies: requirement #7

Requirement #: 22 **Requirement Type**: hidden **Use Case#**: 6
Description: The reservations created will be stored in the database.
Rationale: All the reservations and pre-reservations done must be stored in the database in order to save the needed information to create the connections.
Fit Criterion: The information about the reservations will be stored in the database at the same moment of its request.
Customer Satisfaction: 2 **Customer Dissatisfaction**: 5
Dependencies: requirement #7

Requirement #: 23 **Requirement Type**: evident **Use Case#**: 7
Description: The user will be able to cancel a reservation done previously.
Rationale: The network resources have to be able to being released if the user doesn't wants to reserve a these resources any more.
Fit Criterion: The resources will be released at the same moment of the request.
Customer Satisfaction: 4 **Customer Dissatisfaction**: 5
Dependencies: requirement #1, #6, #7 **Conflicts**:
Supporting Materials:

Requirement #: 24 **Requirement Type**: hidden **Use Case#**: 7
Description: The reservations cancelled by the user will be drop from the database.
Rationale: If a user doesn't want to keep a reservation anymore, it is not necessary to have the information saved in the database.
Fit Criterion: The information of the reservation will be drop from the database at the same moment of the cancellation request.
Customer Satisfaction: 2 **Customer Dissatisfaction**: 3
Dependencies: requirement #23

Requirement #: 25 **Requirement Type**: hidden **Use Case#**: 7
Description: When a user requests to cancel a reservation, the reservation indicated must exist.
Rationale: The identifier given by the user must exist in order to cancel the reservation.
Fit Criterion: -
Customer Satisfaction: 2 **Customer Dissatisfaction**: 3
Dependencies: requirement #23

Requirement #: 26 **Requirement Type**: hidden **Use Case#**: 7
Description: When a user requests to cancel a reservation, the connections active will be deleted physically.
Rationale: If there are some connections active of the reservation requested to be cancelled, these connections will be deleted.
Fit Criterion: The connections will be deleted at the moment.
Customer Satisfaction: 2 **Customer Dissatisfaction**: 4
Dependencies: requirement #23

Requirement #: 27 **Requirement Type**: evident **Use Case#**: 8
Description: The user will be able to ask for the status of a service of a reservation.
Rationale: It is interesting to know the status of the reservations done (pending, active, completed, cancelled_by_system, cancelled_by_user, ...).
Fit Criterion: The user can request the status of a reservation whenever he wants.
Customer Satisfaction: 4 **Customer Dissatisfaction**: 4
Dependencies: requirement #1, #3

Requirement #: 28 **Requirement Type**: hidden **Use Case#**: 8
Description: When a user request the status of a reservation, the services indicated must exist.
Rationale: The identifier of the services requested must exist in order to retrieve the information about the status of these services.
Fit Criterion: -
Customer Satisfaction: 4 **Customer Dissatisfaction**: 4
Dependencies: requirement #27

Requirement #: 29 **Requirement Type**: evident **Use Case#**: 8
Description: When a user requests the status of a reservation, he can indicate some services of the reservation or anyone.
Rationale: If the user does not indicate any service, the status of all the services of the reservation will be returned; otherwise, only the status of the services requested will be returned.
Fit Criterion: -
Customer Satisfaction: 2 **Customer Dissatisfaction**: 2
Dependencies: requirement #27

Requirement #: 30 **Requirement Type**: evident **Use Case#**: 9
Description: The user will be able to cancel a job.
Rationale: It is necessary to have the possibility to cancel a group of pre-reservations done.
Fit Criterion: The user can request to cancel a job whenever he wants.
Customer Satisfaction: 3 **Customer Dissatisfaction**: 4
Dependencies: requirement #1, #4, #8

Requirement #: 31 **Requirement Type**: hidden **Use Case#**: 9
Description: The job cancelled by the user will be drop from the database and also its pre-reservations.
Rationale: The information of the job and the pre-reservation will be deleted because it is not necessary to keep it in the database any more.
Fit Criterion: -
Customer Satisfaction: 2 **Customer Dissatisfaction**: 3
Dependencies: requirement #30

Requirement #: 32 **Requirement Type**: hidden **Use Case#**: 9
Description: When a user requests to cancel a job, the job indicated must exist.
Rationale: The identifier must exist in order to cancel the job requested.
Fit Criterion: -
Customer Satisfaction: 2 **Customer Dissatisfaction**: 4
Dependencies: requirement #30

Requirement #: 34 **Requirement Type**: hidden **Use Case#**: 10
Description: When a user requests to complete a job, the job indicated must exist.
Rationale: The identifier must exist in order to complete the job requested.
Fit Criterion: -
Customer Satisfaction: 2 **Customer Dissatisfaction**: 4
Dependencies: requirement #33

Requirement #: 35 **Requirement Type**: hidden **Use Case#**: 10
Description: When a job is completed, the pre-reservations will be cancelled (set the status attribute to 'cancelled_by_system') if the start time requested is earlier than the actual date.
Rationale: If the start time of the reservations has passed, these pre-reservations cannot be activated anymore, so they must be marked as cancelled.
Fit Criterion: -
Customer Satisfaction: 3 **Customer Dissatisfaction**: 4
Dependencies: requirement #33

Requirement #: 36 **Requirement Type**: evident **Use Case#**: 11
Description: The user will be able to activate a service of a reservation that has the automatic activation set as false.
Rationale: If a reservation has been created with NO automatic activation, the user has to be able to activate manually this reservation.
Fit Criterion: The connections of the reservation will be activated as soon as possible from the actual time.
Customer Satisfaction: 4 **Customer Dissatisfaction**: 4
Dependencies: requirement #1, #3, #7

Requirement #: 37 **Requirement Type**: evident **Use Case#**: 11
Description: When the user requests to activate a service, the service indicated must exist and the status of this service must be 'pending'.
Rationale: The service has to exist to be activated and its status as 'pending', because it the status is 'cancelled' this service cannot be activated anymore.
Fit Criterion: -
Customer Satisfaction: 2 **Customer Dissatisfaction**: 4
Dependencies: requirement #36

Requirement #: 38 **Requirement Type**: evident **Use Case#**: 11
Description: The user can't activate a pre-reservation.
Rationale: A pre-reservation will be never activated. To activate a pre-reservation the job must be completed before.
Fit Criterion: -
Customer Satisfaction: 2 **Customer Dissatisfaction**: 2
Dependencies: requirement #36

Requirement #: 39 **Requirement Type**: hidden **Use Case#**: 12
Description: A service of a reservation will be activated when its start time arrives.
Rationale: If a reservation has been created with the automatic activation requested, the system automatically create the connections of the service requested.
Fit Criterion: The connections of the service will be activated at the start time requested in the creation of the reservation.
Customer Satisfaction: 2 **Customer Dissatisfaction**: 5
Dependencies: requirement #7

Requirement #: 40 **Requirement Type**: hidden **Use Case#**: 13
Description: A service of a reservation will be completed when its end time arrives.
Rationale: When the end time of a service arrives (start time + duration requested) the system deletes the connection of the service.
Fit Criterion: The connections of the service will be deleted at the end time requested in the creation of the reservation.
Customer Satisfaction: 2 **Customer Dissatisfaction**: 5
Dependencies: requirement #7

Requirement #: 41 **Requirement Type**: hidden **Use Case#**: 14
Description: The net is dynamic.
Rationale: Nodes, Endpoints and Links can be added or deleted from the net.
Fit Criterion: -
Customer Satisfaction: 3 **Customer Dissatisfaction**: 5
Dependencies:

4.5.2. Non Functional Requirements

The non functional requirements are characteristics or properties that define how the system must work.

Following you can see the description of the non functional requirements.

Requirement #: 42 **Requirement Type**: hidden **Use Case#**:
Description: The response time of the system must be as fast as possible.
Rationale: If the response time is too big, the user won't be satisfied of the system and won't use it anymore.
Fit Criterion: The response time of each operation won't be more than 30 seconds each one.
Customer Satisfaction: 2 **Customer Dissatisfaction**: 5
Dependencies:

Requirement #: 43 **Requirement Type**: evident **Use Case#**:
Description: Only the users with privileges will be able to interact with the system.
Rationale: As the users are working with physical networks, and the resources are shared, it is important to restrict the access in order to protect the network resources.
Fit Criterion: Only the authorized users will be able to operate with the system
Customer Satisfaction: 4 **Customer Dissatisfaction**: 5
Dependencies:

Requirement #: 44 **Requirement Type**: hidden **Use Case#**:
Description: The product must be OS independent.
Rationale: This product must works with any operating system in order to be able to the maximum number possible of users, so it will be implemented in java.
Fit Criterion: -
Customer Satisfaction: 4 **Customer Dissatisfaction**: 5
Dependencies:

Requirement #: 45 **Requirement Type**: hidden **Use Case#**:
Description: The system must be robust.
Rationale: If the system not tolerates possible fails, the user won't be satisfied.
Fit Criterion: If the service falls down, when it turns up the following cases must be checked:

a. If a reservation was active and its end time hasn't arrived when the system turns up again, the scheduler must be programmed to finish connections of the reservation at its end time.
b. If a reservation was active and its end time has arrived when the system turns up again, the connections of the reservation must be deleted. And the status is marked as 'cancelled_by_system'.
c. If a reservation was pending and its end time has arrived when the system turns up again, the system must mark the status as 'cancelled_by_system'.
d. If a reservation was pending and its end time hasn't arrived when the system turns up again, the system creates all the connections and programs the scheduler to its end time.
e. If it was a pre-reservation and the start time has arrived the system must mark the status as 'cancelled_by_system'.

Customer Satisfaction: 5 **Customer Dissatisfaction**: 5
Dependencies:

Requirement #: 47 **Requirement Type**: hidden **Use Case#**:
Description: The system must be scalable and extensible.
Rationale: New functionalities can be added without making changes to the functionalities done.
Fit Criterion: -
Customer Satisfaction: 3 **Customer Dissatisfaction**: 3
Dependencies:

Requirement #: 47 **Requirement Type**: hidden **Use Case#**:
Description: The data stored in the database must be reliable and correct.
Rationale: If the data stored in the database is not reliable and correct, the user won't be satisfied and the system won't work properly.
Fit Criterion: -
Customer Satisfaction: 2 **Customer Dissatisfaction**: 5
Dependencies:

Requirement #: 48 **Requirement Type**: hidden **Use Case#**:
Description: The system must be reusable.
Rationale: The product must be reusable in order to the maximum number of users could use it.
Fit Criterion: -
Customer Satisfaction: 2 **Customer Dissatisfaction**: 3
Dependencies:

Requirement #: 49 **Requirement Type**: evident **Use Case#**:
Description: The system must works with the following machines: Nortel OPTera Metro 5200, Nortel Optical Multiservice Edge 6500, Nortel HDXc, Cisco ONS 15454.
Rationale: The system must work with this machines in order to make the demonstration mentioned with success.
Fit Criterion: -
Customer Satisfaction: 3 **Customer Dissatisfaction**: 5
Dependencies:

5. PLANNING

The planning shows in detailed the temporal structure of the different tasks done during the development of the project.

5.1. INITIAL PLANNING

Once the requirements analysis is done, the initial planning of the project can be carried out. The project has been planed following a spiral model divided in two phases (Phase 1 and Phase2).

- Phase 1: In the first phase the basic features will be developed. A specification, design and implementation of these features will be done and they also will be tested. But firstly, a period of environment familiarization will be needed; this period is included in this phase. The basic features developed in this phase are the following ones:
  - Get Endpoints
  - Get Reservations
  - Is Available
  - Create Reservation
  - Cancel Reservation
  - Activate automatically
  - Complete service

  In this phase a static network that won't change is considered. During the first phase, the reservations will be only "fixed" reservations.
- Phase 2: In this phase the advanced features will be implemented. Also the "deferrable" and "malleable" reservations will be implemented. These advanced features are the following ones:
  - Get Jobs
  - Complete Job
  - Cancel Job
  - Activate
  - Reconfigure system

  In this phase it is possible to insert or delete links, endpoints and nodes of the network. The requirements will be reviewed and also more requirements will be added. Finally, the service described in this thesis will be integrated into the UCLP system and the PHOSPHORUS IST FP6 project.

This planning has been divided in these two phases because on 13th November there has been a demonstration of the Phosphorus in the SC07 congress, and the features requested for this demonstration are the ones mentioned in the first phase; thus, the ones that must work properly first.

5.2. INITIAL PLANNING: PHASES OF THE PROJECT

In this section the phases mentioned before are described in detail and also it is shown the initial calendar that begins on 5th February 2007.

5.2.1. Phase 1

Following there is the list of the sub phases established for this phase.

1. Familiarization with the development environment
2. Selection of the requirements for the basic functionalities
3. Specification

4. Design
5. Implementation
6. Testing

I. Familiarization with the development environment

As the main goal of this project is to integrate the UCLP system into the PHOSPHORUS IST FP6 project, and these projects cover many more areas than the computer one and enter non-trivial concepts related to the networks and communications world, it has been considered a necessary stage of familiarity with the technologies associated with the project, that in many cases were unknown. Although I have experience in the Java language and I am familiar with the web technologies, it will be necessary to invest time in understanding and go deep in the technologies required for this project, not only in the field of programming (JSP's, Servlet's...), but also working with Web servers (Tomcat) and database tools (Hibernate, PostgreSQL).

These tasks have been defined as one sub phase included in the first phase, but it is expected to be present in the other sub phases and in the other phase.

II. Selection of requirements

In this sub phase, a selection of the requirements of the basic features will be done. From all the requirements defined at the beginning of the project, in this phase only the ones that relate to the basic features mentioned previously will be implemented.

III. Specification

Once the requirements are defined, the first phase continues with the normal procedure for the development of a computer system: Specification, Design, Implementation and Testing.

IV. Design

During the design phase, the first important decisions that will affect the other parts of the development process will be taken in order to accomplish the requirements specified in the last sub phase. A first design of the database will be performed; this design won't be the definitively one because the specifications of the product may change during the development process.

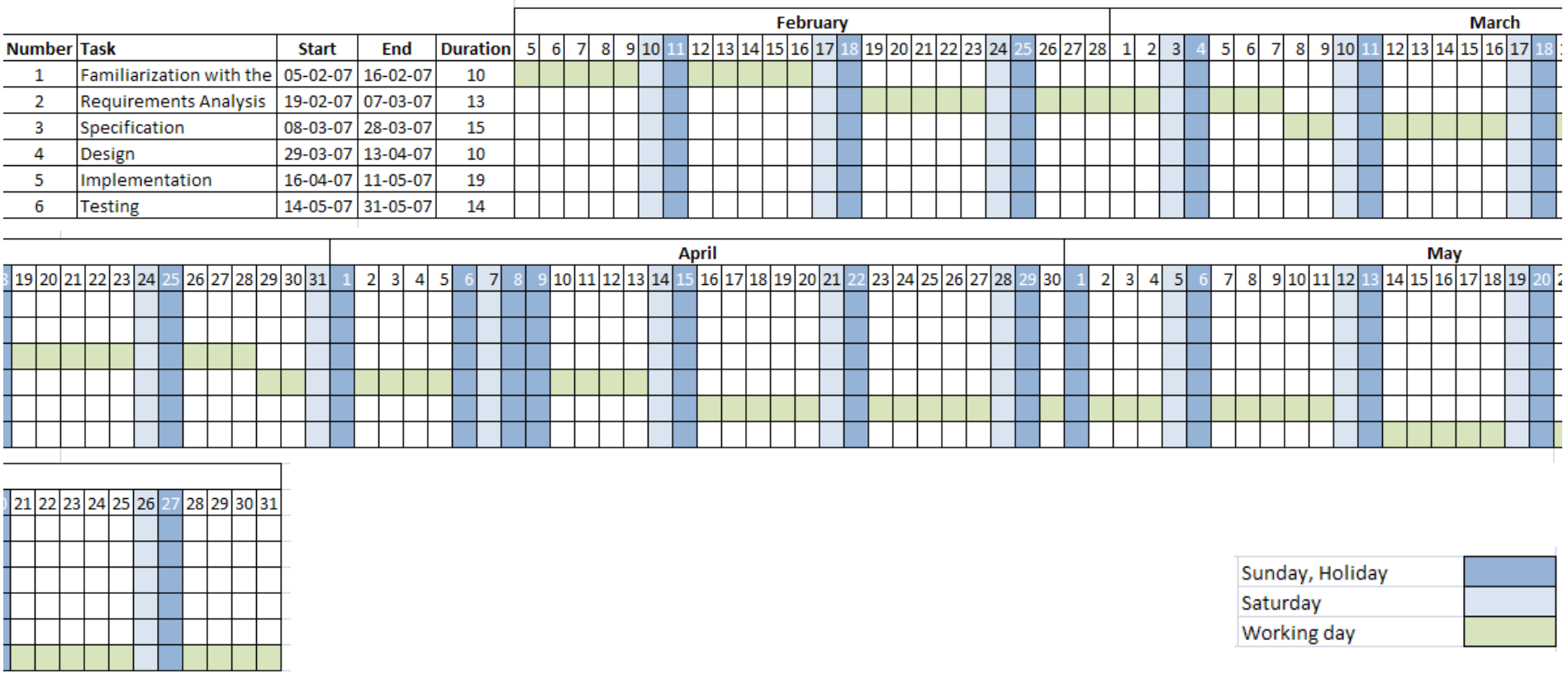
V. Implementation

The implementation of a first prototype will be done. This phase will go on until the end of the project.

VI. Testing

A GUI will be developed to test the basic functionalities performed in this phase. This GUI will be extended in the second phase and used to test the other functionalities.

In the following table the planning calendar for the first phase is shown.



Number	Task	Start	End	Duration
1	Familiarization with the	05-02-07	16-02-07	10
2	Requirements Analysis	19-02-07	07-03-07	13
3	Specification	08-03-07	28-03-07	15
4	Design	29-03-07	13-04-07	10
5	Implementation	16-04-07	11-05-07	19
6	Testing	14-05-07	31-05-07	14

Table 1. Initial planning of the first phase

5.2.2. Phase 2

The following list shows the sub phases established for this second phase of the development process.

1. Review requirements and add new
2. Specification
3. Design
4. Implementation
5. Testing

I. Review requirements and add new

After the first phase, the selected requirements will be reviewed and the ones that have not been selected will be specified and developed in this phase. New requirements for the advanced functionalities will be added to the list. If the specification of the product changes during the development process, the new requirements that will satisfy these changes will be also added to the list of requirements.

At this point, the possibility of changing the specification in order to integrate the service described in this project into the UCLP system and the PHOSPHORUS IST FP6 project will be regarded.

II. Specification

Once the requirements are defined, the second phase will continue with the normal procedure for the development of a computer system: Specification, Design, Implementation and Testing.

III. Design

The design of the first phase will be reviewed. Changes may occur due to the addition of the advanced functionalities. However, these changes will be added to the design and implemented regarding all the considerations of the requirements and the specification of this second phase.

IV. Implementation

The functionalities added and the changes regarding the first phase will be implemented. The service will be adapted to communicate with the UCLP system and with the PHOSPHORUS IST FP6 project.

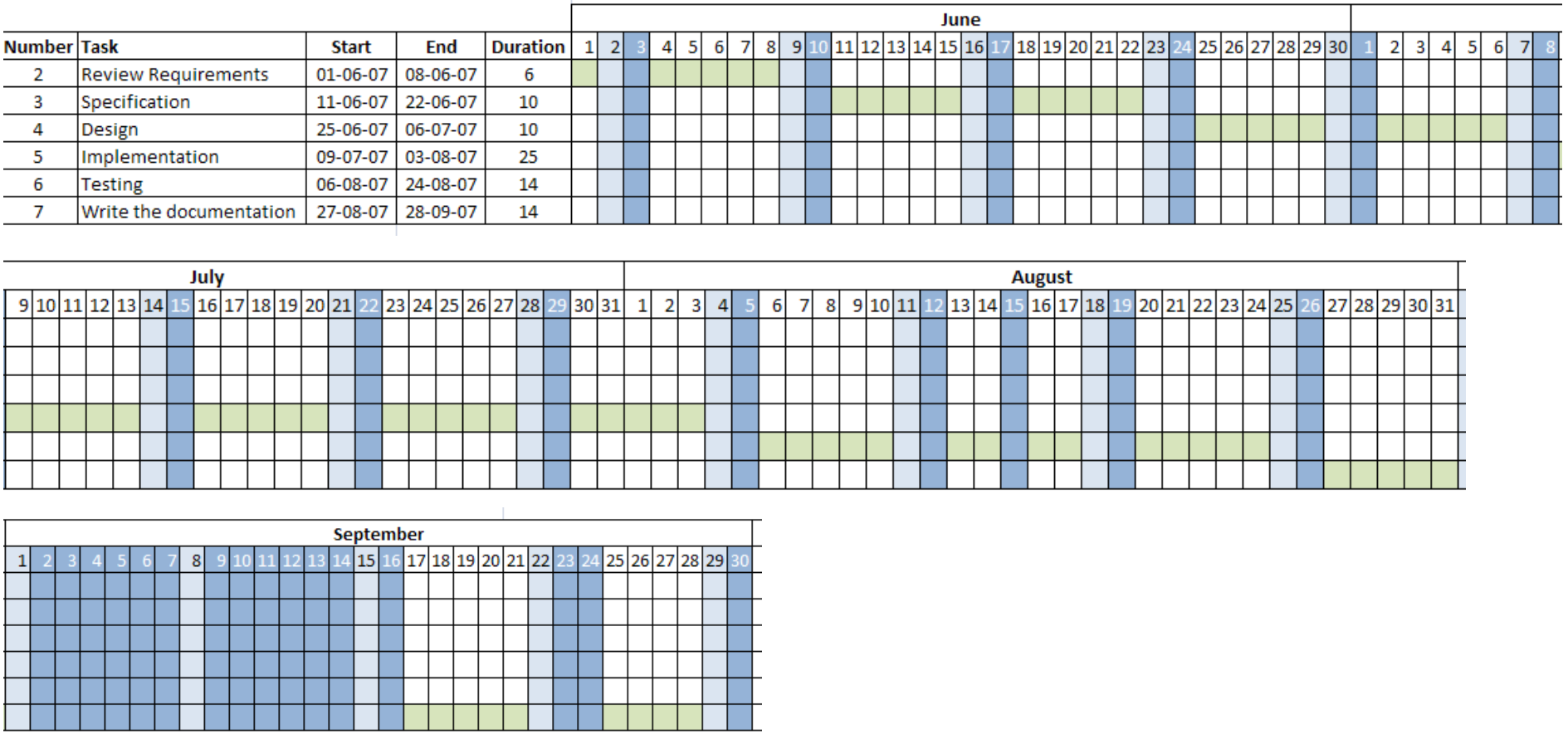
V. Testing

To test the service integrated with the UCLP, the GUI developed in the first phase will be extended to support the new features and it will be used to test them.

To test the UCLP integrated in the PHOSPHORUS IST FP6 project thanks to the ARS there will be two demonstrations:

- Demonstration of the Phosphorus in the SC07 congress on 13th November.
- Demonstration of the Phosphorus in the EC Review on 13th December.

In the following table the planning calendar for the second phase is shown.



Number	Task	Start	End	Duration
2	Review Requirements	01-06-07	08-06-07	6
3	Specification	11-06-07	22-06-07	10
4	Design	25-06-07	06-07-07	10
5	Implementation	09-07-07	03-08-07	25
6	Testing	06-08-07	24-08-07	14
7	Write the documentation	27-08-07	28-09-07	14

Table 2. Initial planning of the second phase

5.3. Revision of Planning

This section describes the revision of planning made because of some changes of the project that have occurred due to some unexpected cases that are detailed in the next sections.

5.3.1. Description and Measures

In the specification and implementation of the PHOSPHORUS IST PF6 project there have been some delays, as normal delays that occur on projects with a high level of participation and interaction. These delays have made the PHOSPHORUS IST PF6 project drag on more than planned, and consequently, also the project described in this documentation.

For this reason and to accomplish with de desired quality of the product, the planning has been modified introducing a 3rd phase. The main change is that no deferrable and malleable reservations are considered at the moment; only the fixed ones will be performed. This decision has been taken because for the demonstrations no deferrable and malleable reservations will be required and because there were other priorities and there was not enough time to perform everything. Therefore, this one is the only requirement suppressed from the project at the moment. With these changes, after the second phase, the ARS will be connected to the UCLP system; and after the third phase, the UCLP will be integrated in the PHOSPHORUS IST FP6 project through the ARS. Some changes in the specification of the ARS will be done due to the changes done in the specification of the PHOSPHORUS IST FP6 project.

5.3.2. Final Planning

In short, the three phases can be summed up as follows:

1. Defining requirements, specification, design, implementation and Testing of the ARS. The net is considered static. The basic functionalities developed are the following ones:
    a. Get Endpoints
    b. Get Reservations
    c. Is Available
    d. Create Reservation
    e. Cancel Reservation
    f. Activate automatically
    g. Complete Service
2. Reviewing and adding requirements, reviewing the specification, the design and the implementation and testing the ARS (advanced functionalities) connected to the UCLP System. The network is still considered static. The advanced functionalities developed are the following ones:
    a. Get Jobs
    b. Complete Job
    c. Cancel Job
    d. Activate
    e. Reconfigure System
3. Reviewing and adding requirements, reviewing the specification, the design and the implementation and testing the integration of the UCLP into the PHOSPHORUS IST FP6 project by means of the ARS. In this third phase the network could be dynamic, the specification may change due to this fact.

It must be said that this planning is a reference, and that during the process of developing a software system is inevitable go back, change and redo some step.

Following, the two tables that show the final planning after having taken the measures described in the preceding section can be seen.

Phase 2

Number	Task	Start	End	Duration
2	Review Requirements	01-06-07	06-06-07	4
3	Specification	07-06-07	15-06-07	7
4	Design	18-06-07	29-07-07	10
5	Implementation	02-07-07	03-08-07	25
6	Testing	06-08-07	31-08-07	19

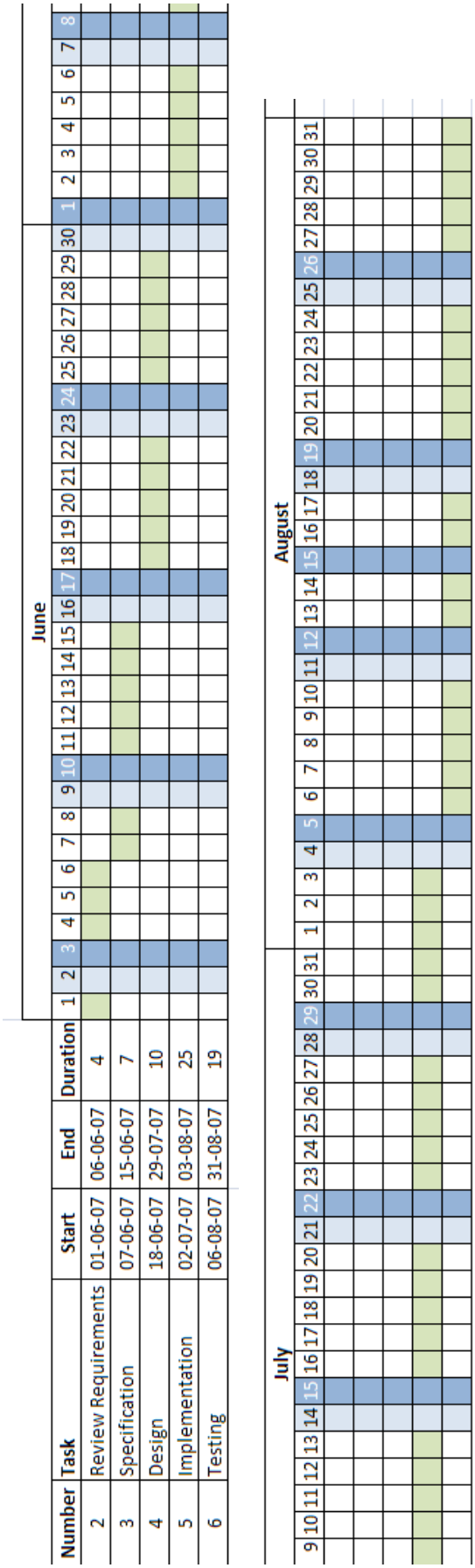


Table 3. Final planning of the second phase

Phase 3

Number	Task	Start	End	Duration
2	Review Requirements	17-09-07	18-09-07	2
3	Specification	19-09-07	21-09-07	3
4	Design	25-09-07	28-09-07	4
5	Implementation	01-10-07	17-10-07	12
6	Testing	18-10-07	02-11-07	11
7	Write the documentation	05-11-07	30-11-07	20

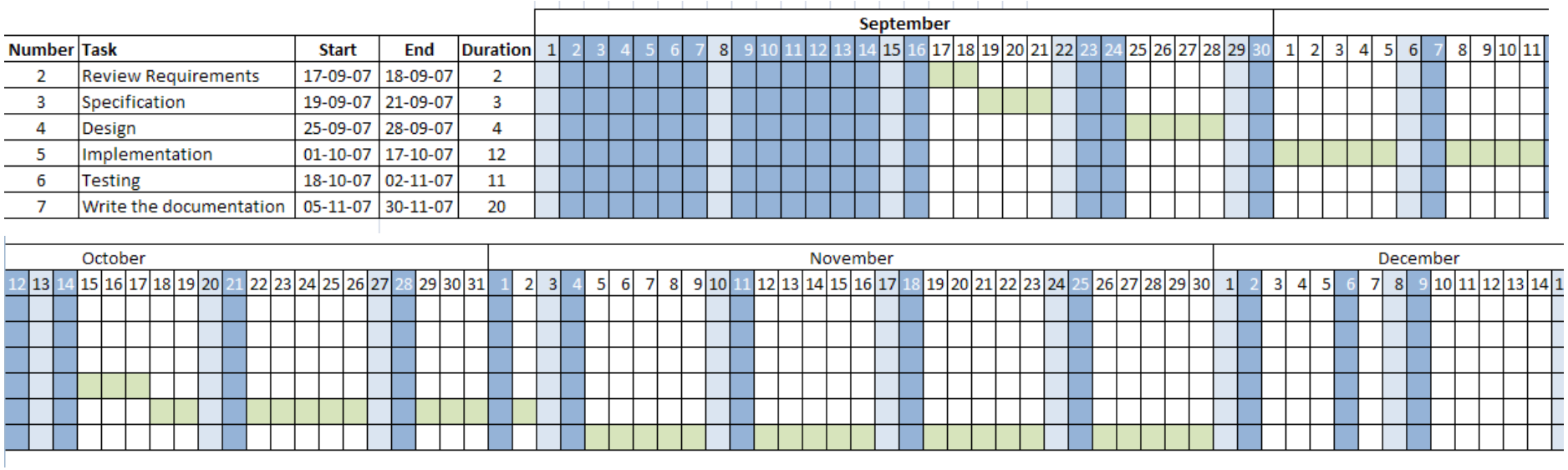


Table 4. Final planning of the third phase

6. Economic Analysis

Like for any software product, for this project an economic analysis has also been performed. Regarding that the software used for the development of this project is free software (JAVA, Tomcat, PostgreSQL, Hibernate, StartUML, Eclipse...), for the economic analysis the number of hours spent in the development of the project will only be considered.

It must be said that the cost will be fictitious because many of the hours spent in the development have been queries, learning and documenting, and this doesn't happen in the commercial products.

To get the closest to reality, two profiles have been defined: analyst and programmer, and the tasks have been distributed as in a software company. Therefore, the analyst is the responsible for the requirements analysis, specification and design, and the programmer carries out the implementation and the tests. The hours dedicated to the familiarization and documentation in the planning have not been counted in the economic analysis, but we must regard that many of the hours of the other tasks have also been dedicated to consulting and documenting.

Thus, regarding the days dedicated to the requirements analysis, specification, design, implementation and testing in the final planning and that an average of 5 hours a day have been worked; paying 16 €/hour to the analyst and 13 €/hour to the programmer, the next count is obtained:

	Analyst	Programmer
Phase 1	38 days	33 days
Phase 2	21 days	44 days
Phase 3	9 days	23 days
Total days profile	68 days	100 days
Total hours profile	340 hours	500 hours
Total € profile	5440 €	7000 €
Total €	**12440 €**	

Table 5. Economic Analysis

7. SPECIFICATION

The specification is the first step in the system engineering process after the analysis of requirements. A specification is an explicit set of requirements to be satisfied by a service. It provides the necessary details about the specific requirements and the description of the behaviour of the external part of the global system from the user's and environment's perspective. This description is done without entering in the details of the solution. A specification includes the use case model, the data conceptual model, the behaviour of the system model and the states model.

This section, and the following ones that belong to the system development process (Specification, Design, Implementation and Testing), contains all the functionalities corresponding to the use cases defined in section 4.4.

7.1. USE CASE MODEL

The purpose of the use case model is to determinate the specification of the requirements found in the analysis of requirements (section 4). In this way, the model determines which the functions of the system are for each actor (section 4.2.2). The use case diagram and the use case list are defined in the section 4.4 because for the requirements analysis the Volere methodology has been followed.

7.2. CONCEPTUAL MODEL

The conceptual model is the representation of the significant concepts (objects) in the domain of the problem. The goal is to describe the information that the system has to store in a persistent way, including the structure of the information and the restrictions of this information.

Two types of management groups are included in the system:

- Reservation Management Group: Contains the operations that refer to the management of the reservations (create, cancel, consult...). The classes that belongs to this group are the following ones:
    - Job
    - Reservation
    - Service
    - Connection
    - FixedReservation
    - DeferrableReservation
    - MalleableReservation
- Topology Management Group: Contains the operations that refer to the management of the topology of the network (get the elements, add, delete...). The classes that belongs to this group are the following ones:
    - Endpoint
    - Link
    - Node

As it has been explained in the 5th section (Planning), in the first and second phases of the planning, a static network has been considered, while in the third phase it is

considered a dynamic one. Thus, the conceptual model designed in the first phase has been changed in the third one for reasons of efficiency that will be explained below.

Firstly, to understand this change, the conceptual model of the first and second phases is shown in the following figure. In this case, the classes of both groups (Reservation and Topology Management) are included in the same class diagram and are related by the associative class ConnectionEndpoint (indicates if one endpoint in one connection is the source endpoint, the target endpoint or just an endpoint of the path).

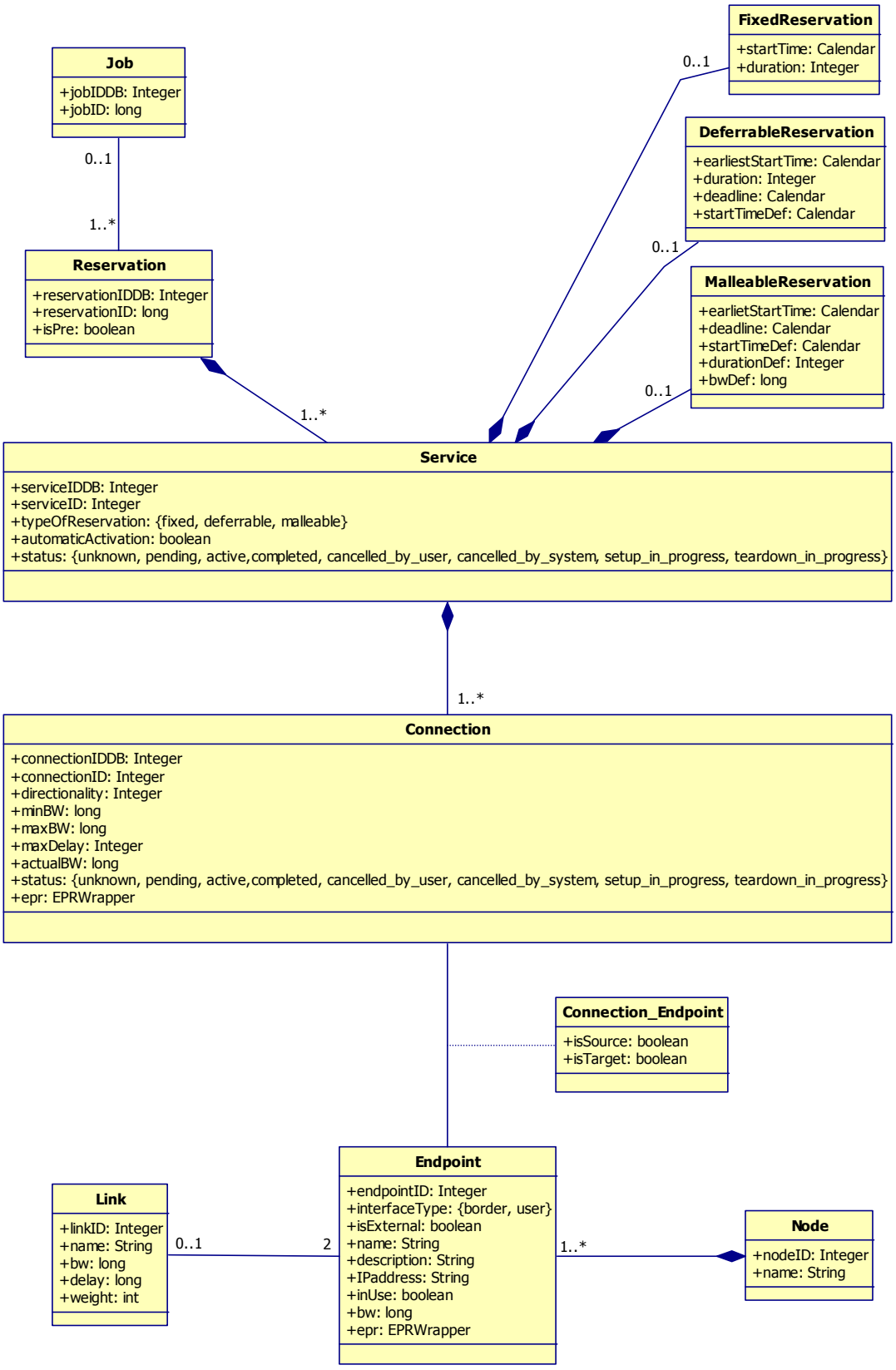


Figure 10. Conceptual Model 1st phase

Textual Restrictions:

- Primary key:
    - Two different nodes can't have the same nodeID.
    - Two different endpoints from the same node can't have the same endpointID.
    - Two different links can't have the same linked.
    - Two different jobs can't have nor the same jobIDDB neither the same jobID.
    - Two different reservations can't have nor the same reservationIDDB neither the same reservationID.
    - Two different services belonging to the same reservation, can't have nor the same serviceIDDB neither the same serviceID.
    - Two different connections belonging to the same service, can't have nor the same connectionIDDB neither the same connectionID.
- The values allowed for the attribute 'directionality' in the class Connection are 0 (unidirectional tree), 1 (bidirectional tree) and 3 (full mesh).
- An endpoint that belongs to a connection can be source, target, or nothing, but no source and target at the same time.

In the third phase of the planning, as the network is dynamic, some decisions in the conceptual model have been taken.

Every time the network changes, the system must be restarted; and every time that the system is started up, the topology network file is retrieved from the UCLP system. These possible changes in the network make so inefficient the fact that the information about the network is persistence, because it is necessary to check if the net has changed, update the network in the database, and also update the paths of the connections if necessary. So, in order to get the maximum benefit of the resources, the topology of the network will be saved in a data structure in the memory. In this way, every time that the service is started up, a new net is created and it is only necessary to check the path of the connections of the reservations. This is more simple to implement and more efficient. Thus, the net is created in the memory as a singleton class. To sum up, as every time the system is started up the topology file is retrieved, it is no necessary to have the net structure stored in the database.

Note that, in this case, the classes from the Reservation Management Group and the ones from the Topology Management Group are not in the same diagram. The union for both groups of classes is the endpointID present in the class ConnectionEndpoint of the Reservation Management group and in the Endpoint class of the Topology Management group. Only the classes of the Reservation Management Group are persistent, the ones of the Topology Management Group no. The textual restrictions are the same than the ones of the first phase of the planning.

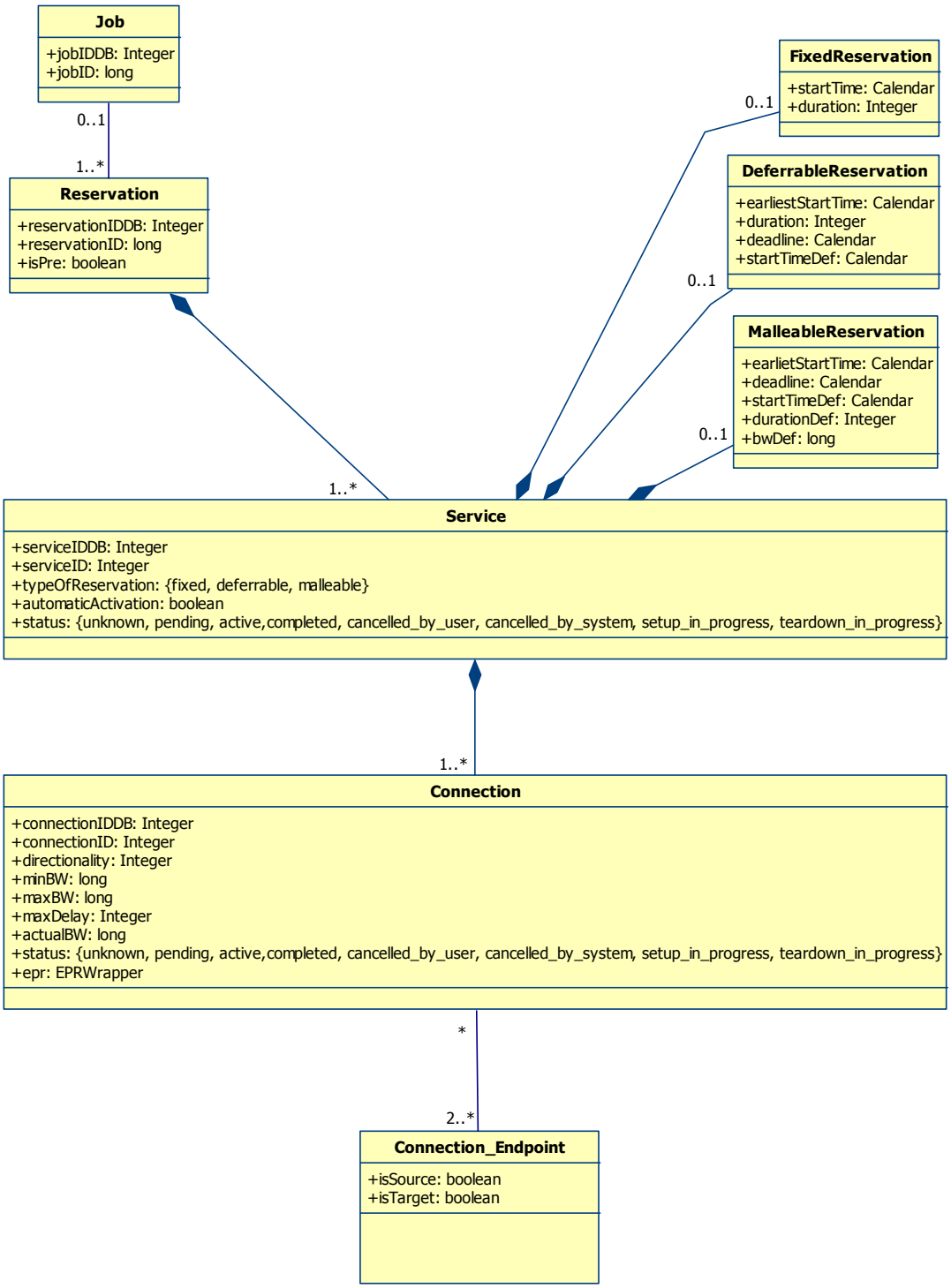


Figure 11. Conceptual Model 2nd phase. Reservation Management

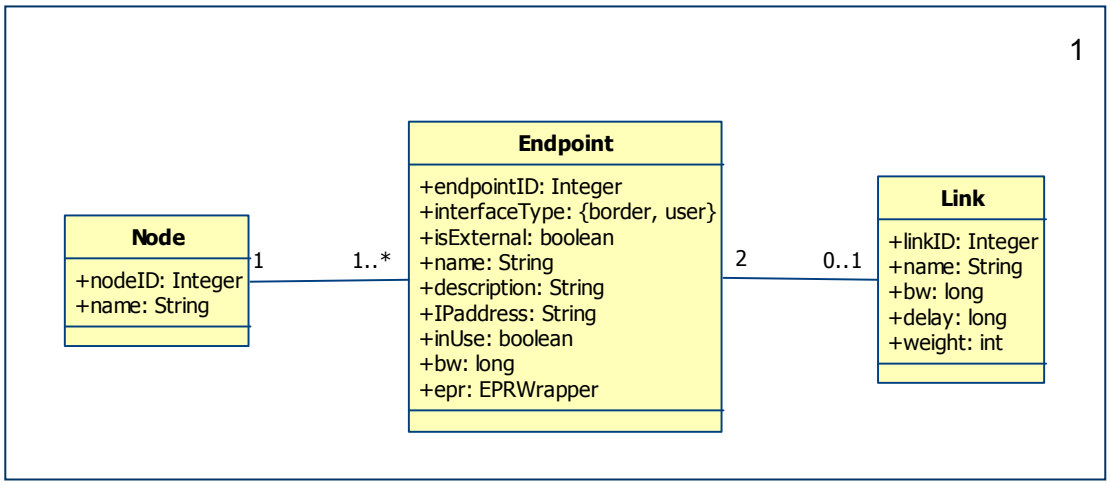


Figure 12. Conceptual Model 2nd phase. Topology Management

7.3. BEHAVIOUR MODEL

Every system developed using the Object Oriented Programming technique, is compounded by a number of classes (objects), and each one has attributes, operations and its own behaviour. The objects communicate each other's calling the functions of the other objects. In the specification of the behaviour of the system it is too early to talk about final classes. So, using an abstracting process, all the classes (objects) will be grouped in a special class called System.

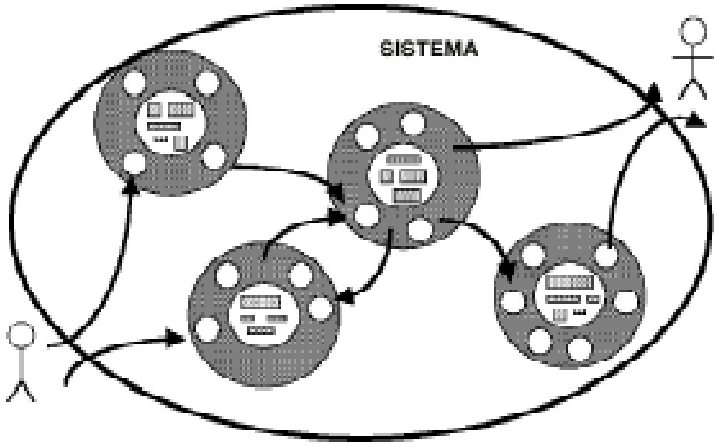


Figure 13. Behaviour Model

In this way, the specification of the behaviour is done with the 'System' behaviour model.

The System behaviour model is compounded by:

- Sequence diagrams of the system: they show the sequence of the events between the actors and the system and identify the operations of the system. Their starting points are the uses cases and its description.
- Operation contracts of the system: they describe the effect of the operations of the system.

7.3.1. Sequence Diagrams

In this section there are the sequence diagrams of the use cases described in section 4.4.2. They are ordered in the same way in order to make easier its comprehension.

The word 'user' will be used to refer to any of the actors of the system (at present these actors can be a Human User or a Grid Middleware, the ones that form part of the PHOSPHORUS Consortium, as it has been explained in the 4th section).

I. Get Features

The user requests the features of the system giving the user identifier and the password. The system authenticates the user and returns the features of the system.

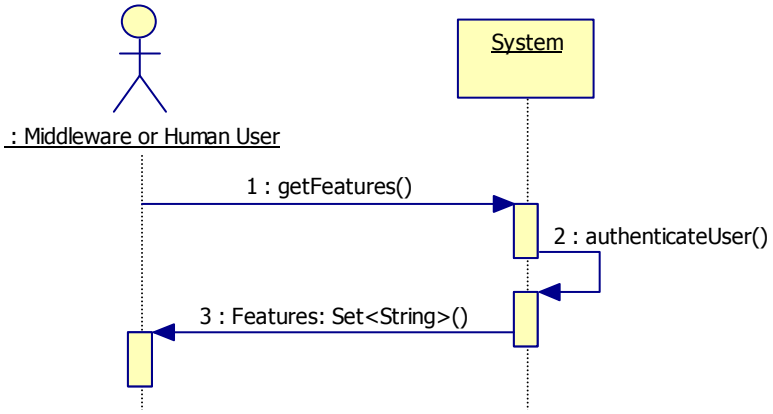


Figure 14. Get Features

II. Get Endpoints

The user requests the endpoints of the network giving the user identifier and the password. The system authenticates the user and returns a list of the endpoints.

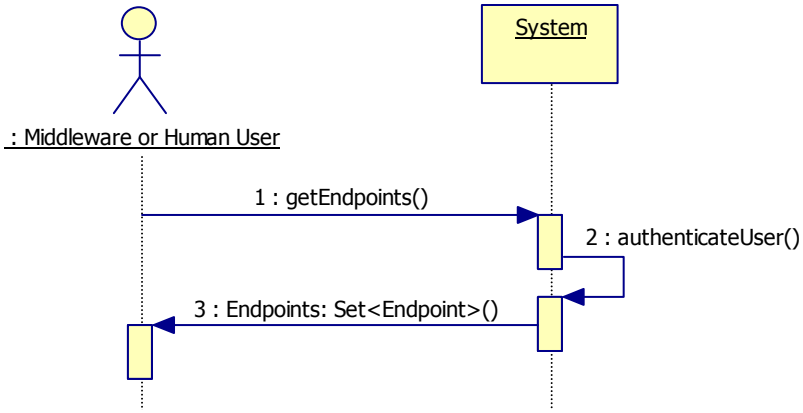


Figure 15. Get Endpoints

III. Get Reservations

The user requests the reservations done giving the user identifier and the password. The system authenticates the user and returns a list of the reservations done in the system.

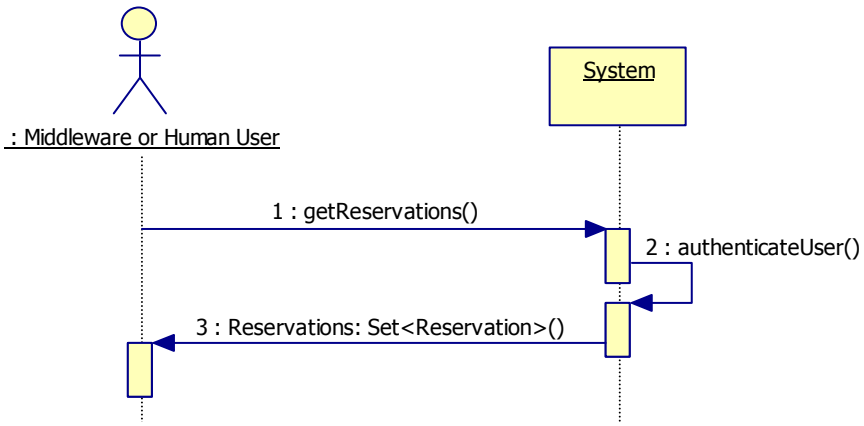


Figure 16. Get Reservations

IV. Get Jobs

The user requests the jobs giving the user identifier and the password. The system authenticates the user and returns a list of the existent jobs in the database.

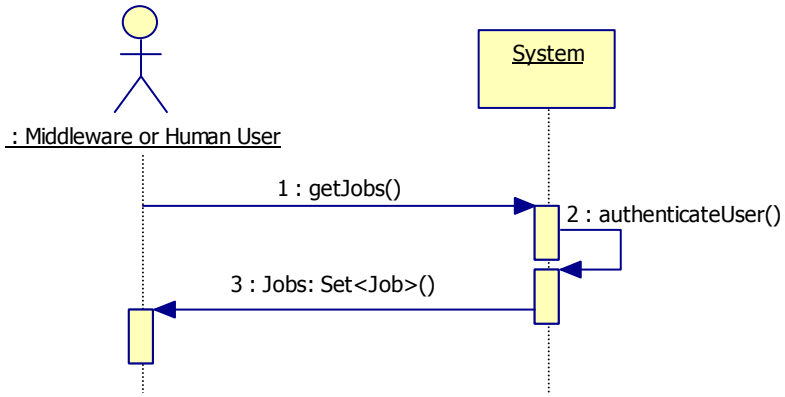


Figure 17. Get Jobs

V. Is Available

The user makes a request of availability giving the user identifier, the password and the resources of the network that he wants to ask for its availability. The system authenticates the user, checks the availability of the resources and returns a response.

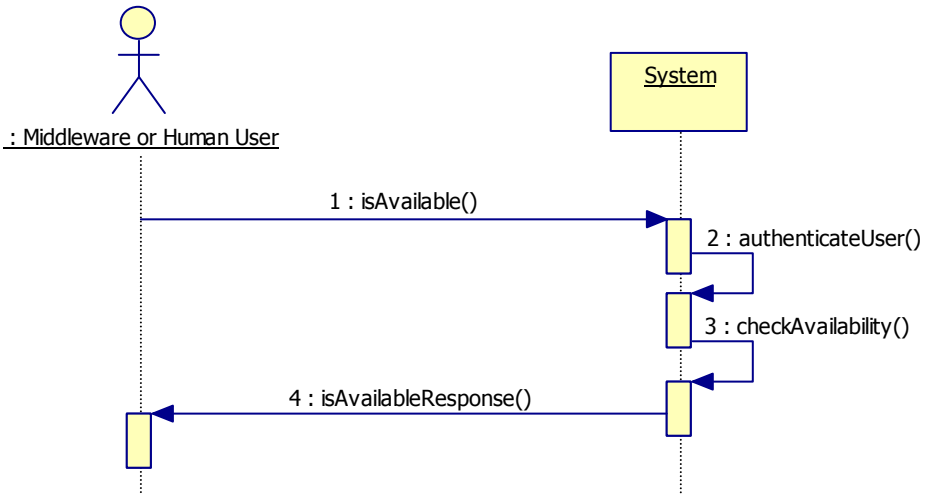


Figure 18. Is Available

VI. Create Reservation

The user wants to create a reservation and gives the user identifier, the password and the resources of the network that he wants to reserve. The system authenticates the user, checks the availability of the resources and if they are available registers the reservation, activates the scheduler and returns the response to the user.

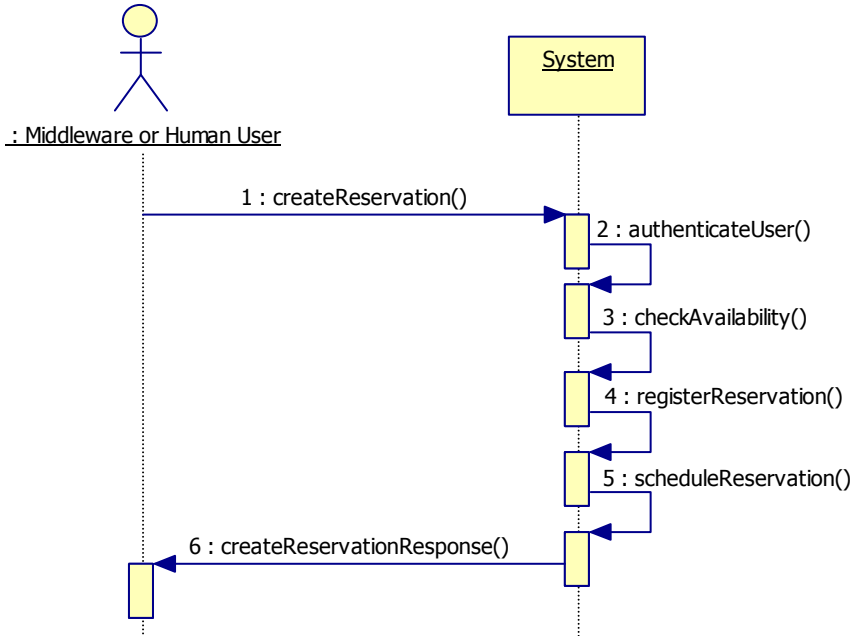


Figure 19. Create Reservation

VII. Cancel Reservation

The user wants to cancel a reservation; he gives the user identifier, the password and the identifier of the reservation he wants to cancel. The system authenticates the user, checks if the reservation exists, and depending on the status of the reservation calls the UCLP system to delete the connections. The system updates the database and returns a response to the user.

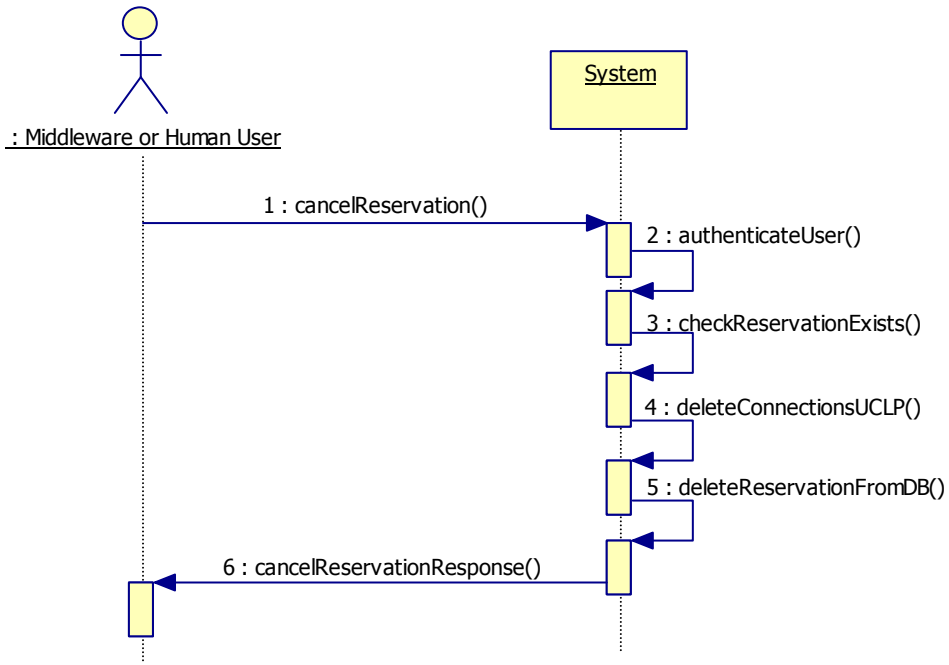


Figure 20. Cancel Reservation

VIII. Get Status

The user wants to know the status of a reservation; he gives the user identifier, the password and the identifier of the reservation that he wants to cancel. The system authenticates the user, checks if the reservation exists and returns the response to the user.

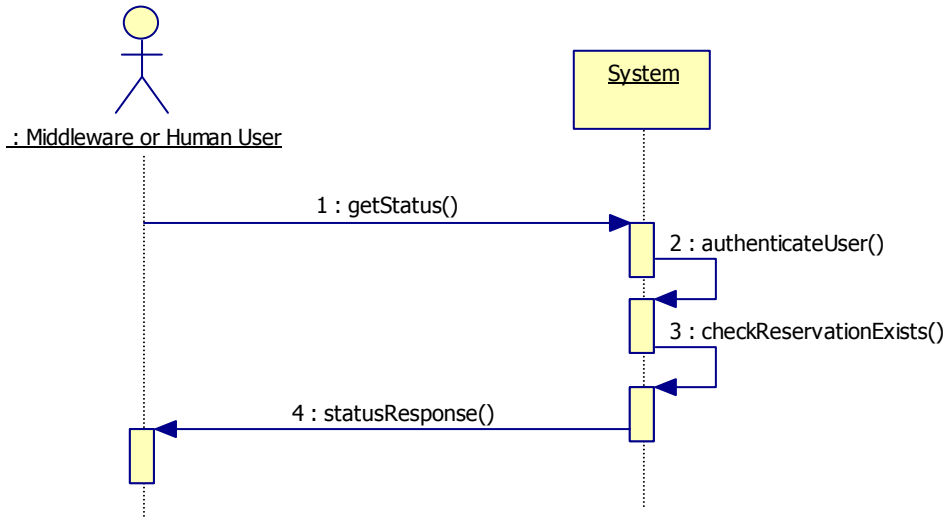


Figure 21. Get Status

IX. Complete Job

The user calls the system to complete a job indicating the user identifier, the password and the identifier of the job to be completed. The system authenticates the user, checks if the job exists, converts all the pre-reservations of the job into permanent reservations, schedules the reservations, deletes the job from the database and returns a response to the user.

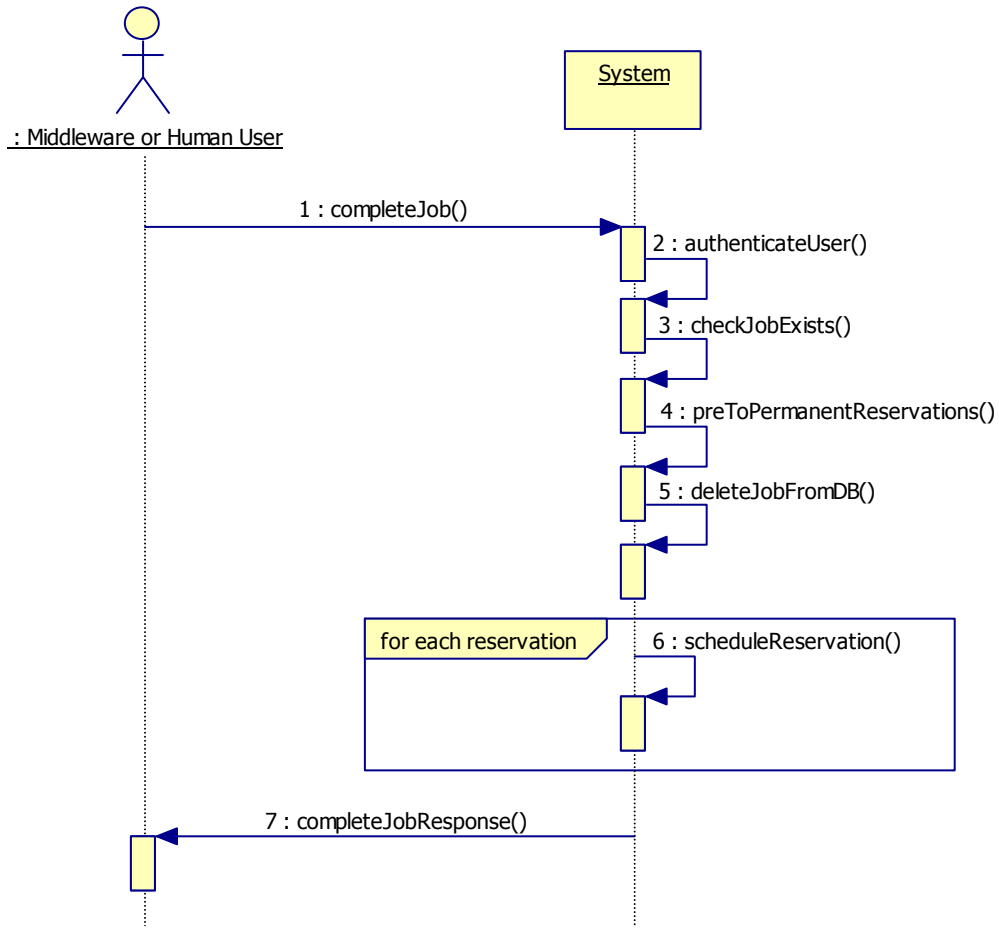


Figure 22. Complete Job

X. Cancel Job

The user calls the system to cancel a job indicating the user identifier, the password and the identifier of the job to be cancelled. The system authenticates the user, checks if the job exists, deletes all the pre-reservations and the job from the database and returns a response to the user.

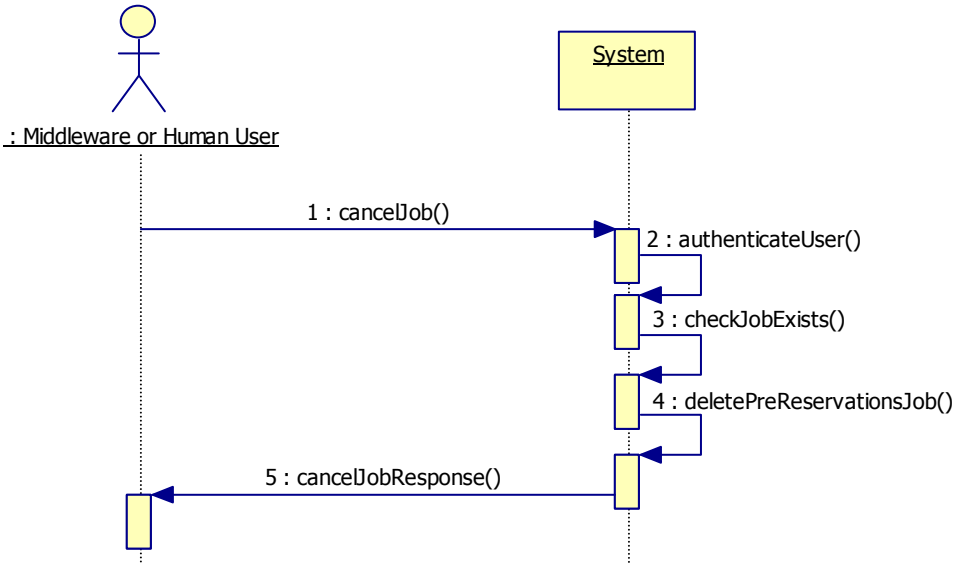


Figure 23. Cancel Job

XI. Activate

The user calls the system to activate a service. The user indicates its identifier, the password, the identifier of the service and the identifier of the reservation to the system. The system authenticates the user, checks if the service exists, calls the UCLP system to activate the connections, updates the database changing the status of the connections and returns a response to the user.

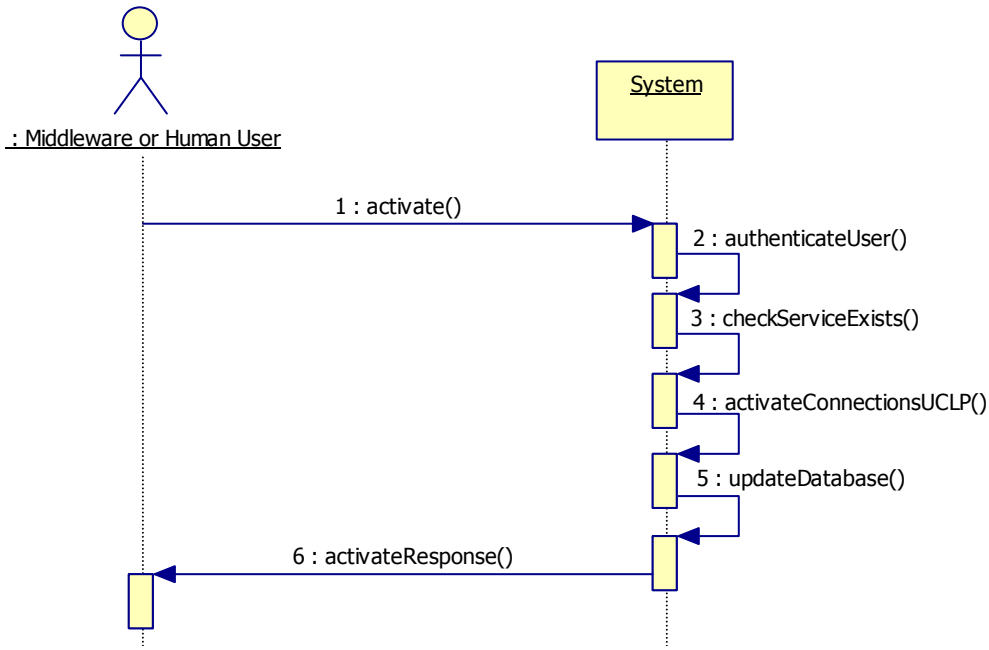


Figure 24. Activate

XII. Activate Automatically

The start time of a service has arrived and the scheduler calls the system to activate a service. It indicates the identifier of the service and of the reservation. The system calls the UCLP system to activate the connections and updates the database changing the status of the connections.

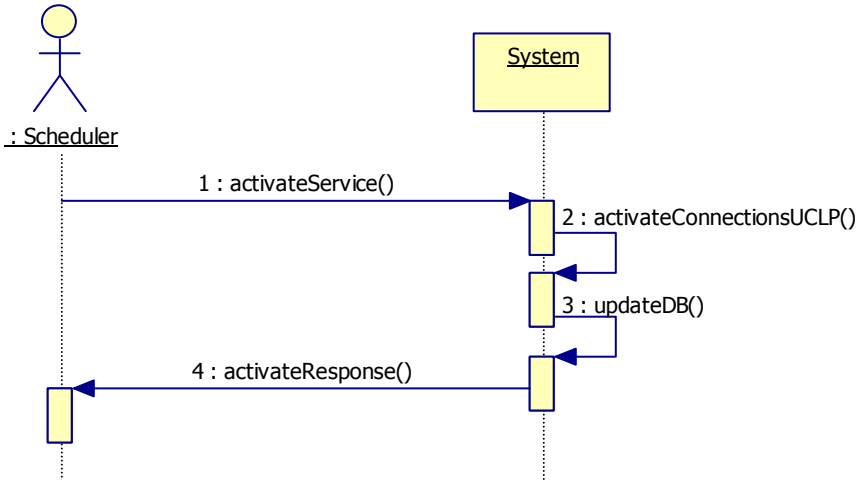


Figure 25. Activate Automatically

XIII. Complete Service

The end time of a service has arrived and the scheduler calls the system to delete the connections of the service. It indicates the identifier of the service and of the reservation. The system calls the UCLP system to delete the connections and updates the database changing the status of the connections.

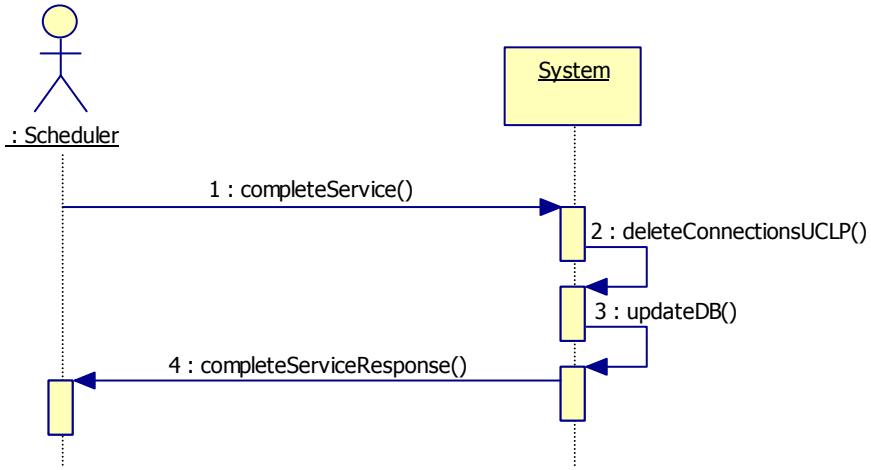


Figure 26. Complete Service

XIV. Reconfigure System

When the service is started up, the system reconfigures the connections stored in the database if it is necessary.

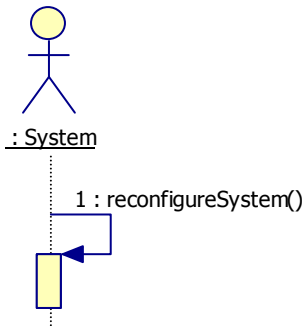


Figure 27. Reconfigure System

7.3.2. Contracts of the Operations

A contract is a document that describes the steps needed so that an operation is to be performed correctly. In the description is more important WHAT will happen than HOW will happen. The contracts are done for each important operation of the system.

Following, the behaviour of the system in terms of outputs generated is described. Preconditions, post conditions and the outputs of the operations are included.

In the following table the relation between the use cases described in the section 4.4.2 (on the right side) and the operations needed to perform them (on the left side) is shown. The objective of this table is to help the comprehension of the contracts and the global working of the system. The use cases of the table are identified by the number that have assigned in the section 4.4.2.

Operation	Use Case (format: number-[name])
authenticateUser	1-Get Features; 2-Get Endpoints; 3-Get Reservations; 4-GetJobs; 5-Is Available; 6-Create Reservation; 7-Cancel Reservation; 8-Get Status; 9-Complete Job; 10-Cancel Job; 11-Activate
getFeatures	1-Get Features
getEndpoints	2
getReservations	3
getJobs	4
isAvailable	5
checkAvailability	5; 6
createReservation	6
registerReservation	6
scheduleReservation	6; 9
cancelReservation	7
checkReservationExists	7; 8
deleteConnectionsUCLP	7; 13
deleteReservationFromDB	7
getStatus	8
completeJob	9
checkJobExists	9; 10
preToPermanentReservations	9
deleteJobFromDB	9
cancelJob	10
deletePreReservationsJob	10
Activate	11
checkServiceExists	11
activateConnectionsUCLP	11; 12
updateDB	11; 12; 13
activateService	12-Activate Automatically
completeService	13-Complete Service

deleteConnectionsUCLP	13
reconfigureSystem	14-Reconfigure System

Table 6. Relation contracts of the operations – use cases

I. Authenticate User

Operation: authenticateUser(String userID, String password)

Semantics: The user wants to perform some operation and the system checks if he has the privileges to do it.

Preconditions: A user with the userID and the password given exists.

Postconditions: The user can interact with the system.

Output: -

II. Get Features

Operation: getFeatures(String userID, String password)

Semantics: If the user is valid, the system returns a list with all the features of the system.

Preconditions: A user with the userID and the password given exists.

Postconditions: -

Output: A lit of the features of the system.

III. Get Endpoints

Operation: getEndpoints(String userID, String password)

Semantics: If the user is valid, the system returns a list with the available endpoints of the network.

Preconditions: A user with the userID and the password given exists.

Postconditions: -

Output: A lit of the available endpoints.

IV. Get Reservations

Operation: getReservations(String userID, String password)

Semantics: If the user is valid, the system returns a list with all the reservations done.

Preconditions: A user with the userID and the password given exists.

Postconditions: -

Output: A lit of the reservations done.

V. Get Jobs

Operation: getJobs(String userID, String password)

Semantics: If the user is valid, the system returns a list with all the jobs existing in the DB.

Preconditions: A user with the userID and the password given exists.

Postconditions: -

Output: A lit of the jobs.

VI. Is Available

Operation: isAvailable(String userID, String password, availabilityReq)

Semantics: If the user is valid, the system checks if the connections requested are available at the time given.

Preconditions: A user with the userID and the password given exists.

Postconditions: -

Output: A message indicating which connections are available.

VII. Check Availability

Operation: checkAvailability(availabilityReq)

Semantics: The system checks if the connections requested are available at the time given.

Preconditions: -

Postconditions: -

Output: A message indicating which connections are available.

VIII. Create Reservation

Operation: createReservation(String userID, String password, reservationRequest)

Semantics: If the user is valid, the system checks if the connections requested are available at the time given, and if they are available, registers the information and schedules the reservation.

Preconditions: A user with the userID and the password given exists and the connections requested are available.

Postconditions: The database contains the information of the reservation created.

Output: A message indicating if the reservation requested has been created or not.

IX. Register Reservation

Operation: registerReservation(reservationRequest)

Semantics: The system stores the reservation with its services and its connections in the database.

Preconditions: -

Postconditions: The database contains the information of the reservation created.

Output: -

X. Schedule Reservation

Operation: scheduleReservation(reservationRequest)

Semantics: The system programs the scheduler in order to activate the reservation at the start time requested.

Preconditions: -

Postconditions: The reservation is scheduled.

Output: -

XI. Cancel Reservation

Operation: cancelReservation(String userID, String password, long reservationID)

Semantics: If the user is valid, the system cancels the connections of the reservations if they are activated. The reservation is deleted from the database.

Preconditions: A reservation with the given identifier exists

Postconditions: The reservation is deleted from the database.

Output: A message indicating if the cancellation has been successful.

XII. Check Reservation Exists

Operation: checkReservationExists(long reservationID)

Semantics: The system checks in the database if there is some reservation with the given identifier.

Preconditions: -

Postconditions: -

Output: A message indicating if the reservation exists or not.

XIII. Delete Connections UCLP

Operation: deleteConnectionsUCLP (long reservationID)

Semantics: The system deletes the connections actives in the UCLP system.

Preconditions: The connections are actives.

Postconditions: The connections are cancelled in the UCLP system

Output: -

XIV. Delete Reservation from DB

Operation: deleteReservationFromDB (long reservationID)

Semantics: The system deletes the reservation from the database.

Preconditions: The reservation exists in the database.

Postconditions: The reservation is deleted from the database.

Output: -

XV. Get Status

Operation: getStatus(String userID, String password, long reservationID, Set<long> servicesID)

Semantics: If the user is valid, the system consults the status of the services requested of the reservation.

Preconditions: The user is valid and the reservation and its services exist in the database.

Postconditions: -

Output: A message indicating the status of the reservation and its services.

XVI. Complete Job

Operation: completeJob(String userID, String password, long jobID)

Semantics: If the user is valid, the system converts all the pre-reservations of the job into permanent reservations.

Preconditions: The user is valid and the job exists in the database.

Postconditions: The job is completed.

Output: A message indicating if the job has been completed successfully.

XVII. Check Job Exists

Operation: checkJobExists(long jobID)

Semantics: The system checks in the database if there is some job with the given identifier.

Preconditions: -

Postconditions: -

Output: A message indicating if the job exists or not.

XVIII. Pre to Permanent Reservations

Operation: preToPermanentReservations (long jobID)

Semantics: The system converts all the pre-reservations associated to the job into permanent reservations and schedules all these new permanent reservations.

Preconditions: -

Postconditions: The new permanent reservations are scheduled and saved in the database.

Output: -

XIX. Delete Job from DB

Operation: deleteJobFromDB (long jobID)

Semantics: The system deletes the job from the database.

Preconditions: The job exists in the database

Postconditions: The job is deleted from the database.

Output: -

XX. Cancel Job

Operation: cancelJob(String userID, String password, long jobID)

Semantics: If the user is valid, the system cancels all the pre-reservations of the job and deletes them and the job from the database.

Preconditions: The user is valid and the job exists in the database.

Postconditions: The job and its pre-reservations are deleted.

Output: A message indicating if the job has been cancelled successfully.

XXI. Delete pre-reservations Job

Operation: deletePreReservations(long jobID)

Semantics: All the pre-reservations associated to the job and the job are deleted from the database.

Preconditions: The job exists in the database.

Postconditions: The job and its pre-reservations are deleted.

Output: A message indicating if the job has been cancelled successfully.

XXII. Activate

Operation: activate(String userID, String password, long reservationID, int serviceID)

Semantics: If the user is valid, the system activates the service. The scheduler to complete the service is programmed.

Preconditions: The service exists in the database.

Postconditions: The service has been activated.

Output: A message indicating if the service has been activated successfully.

XXIII. Check Service Exists

Operation: checkServiceExists(long reservationID, int serviceID)

Semantics: The system checks in the database if there is some service with the given identifier.

Preconditions: -

Postconditions: -

Output: A message indicating if the service exists or not.

XXIV. Activate Connections UCLP

Operation: activateConnectionsUCLP(long reservationID, int serviceID)

Semantics: The system checks the start time requested for the service and if it is valid calls the UCLP system to activate the connections of the service.

Preconditions: The start time requested of the service is valid and the connections of the service are inactive.

Postconditions: The connections of the service have been activated.

Output: A message indicating if the service has been activated successfully.

XXV. Update DB

Operation: updateDB (long reservationID, int serviceID)

Semantics: The system updates the service status, and the status of its connections, changing it into ‘active’.

Preconditions: The service exists in the database.

Postconditions: The status of the service and its connections has been updated in the database

Output: -

XXVI. Activate Service

Operation: activateService(long reservationID, int serviceID)

Semantics: The start time requested for the service has arrived and the system activates the service calling the UCLPs system to create the connections of the service. The scheduler to complete the service is programmed.

Preconditions: The service exists in the database.

Postconditions: The service has been activated.

Output: A message indicating if the service has been activated successfully.

XXVII. Complete Service

Operation: completeService(long reservationID, int serviceID)

Semantics: The end time requested for the service has arrived and the system completes the service. The system calls the UCLP system to delete the connections of the service.

Preconditions: The service exists in the database.

Postconditions: The service has been completed.

Output: A message indicating if the service has been completed successfully.

XXVIII. Delete Connections UCLP

Operation: deleteConnectionsUCLP(long reservationID, int serviceID)

Semantics: The system calls the UCLP system to delete the connections of the service.

Preconditions: The connections of the service are active.

Postconditions: The connections of the service have been cancelled.

Output: A message indicating if the service has been completed successfully.

XXIX. Reconfigure System

Operation: reconfigureSystem()

Semantics: The connections done are reconfigured depending on if the net has been changed or if the start and end time have arrived or not.

Preconditions: -

Postconditions: The connections are reconfigured.

Output: -

7.4. STATES MODEL

This step is the last one in the specification of a software system. The purpose of this section is to create the states diagrams for the use cases defined in the section 4.4. By means of the state diagrams it is described the events states sequence that could

happen in the real world, they show how the state of the objects changes along the time during its life in response to the stimulus received and its responses.

7.4.1. States Diagrams

In this section the state diagrams of each use case are shown.

I. Get Features

It can be seen how from the initial state, when the system receives a getFeatures request, the features of the system are shown.

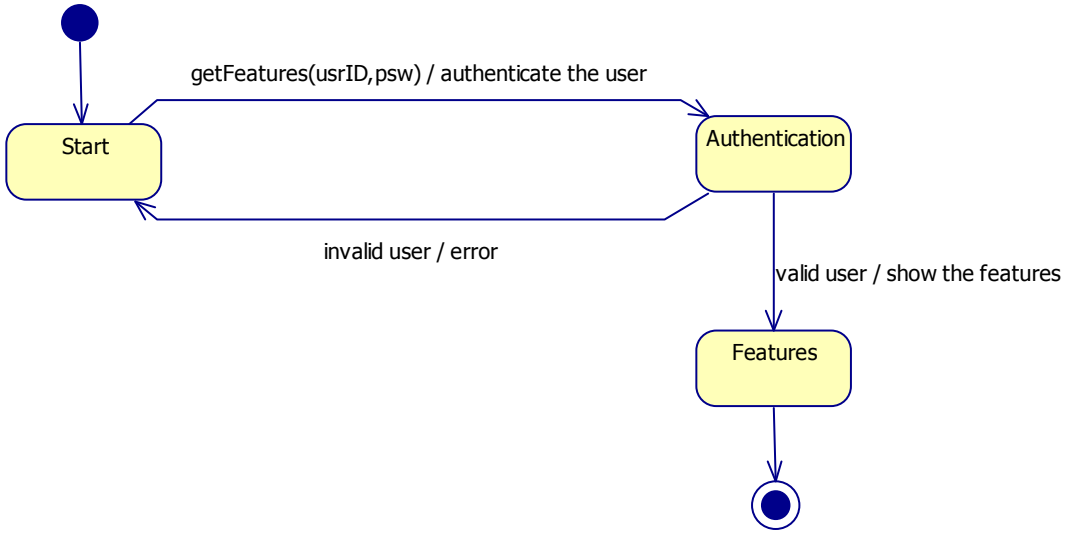


Figure 28. States Diagram: Get Features

II. Get Endpoints

It can be seen how from the initial state, when the system receives a getEndpoints request, the endpoints of the network are shown.

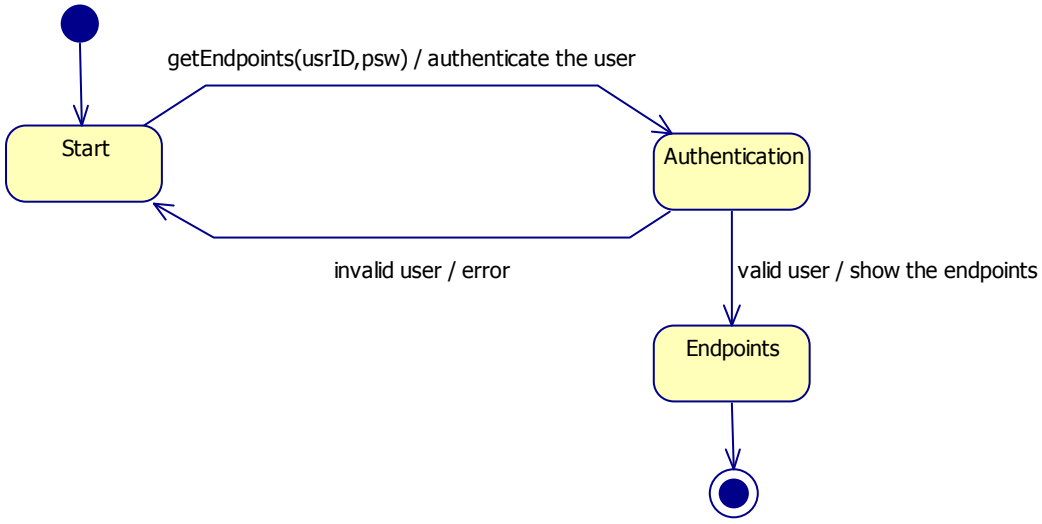


Figure 29. States Diagram: Get Endpoints

III. Get Reservations

It can be seen how from the initial state, when the system receives a getReservations request, the reservations done are shown.

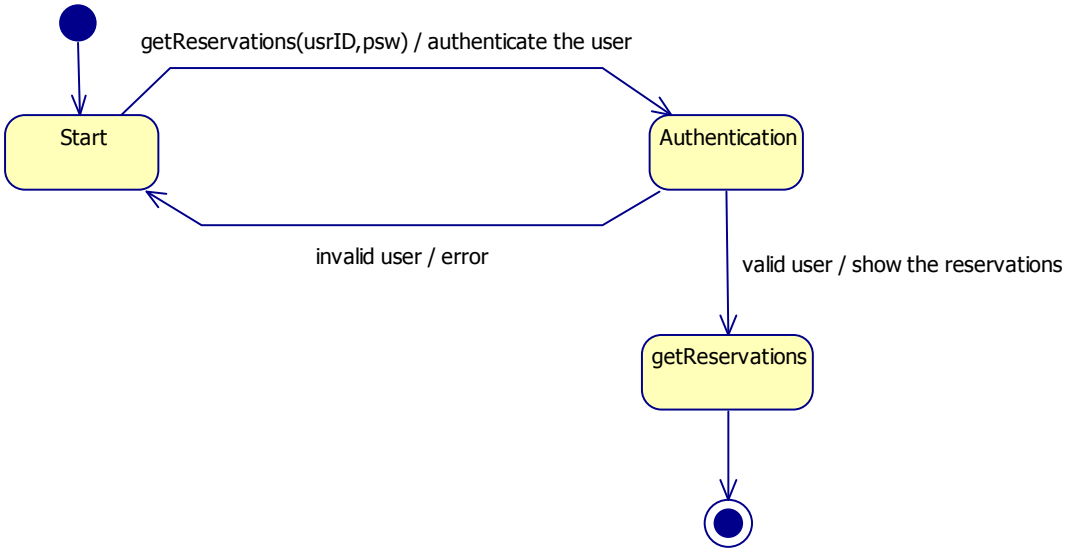


Figure 30. States Diagram: Get Reservations

IV. Get Jobs

It can be seen how from the initial state, when the system receives a getJobs request, the jobs existing in the database are shown.

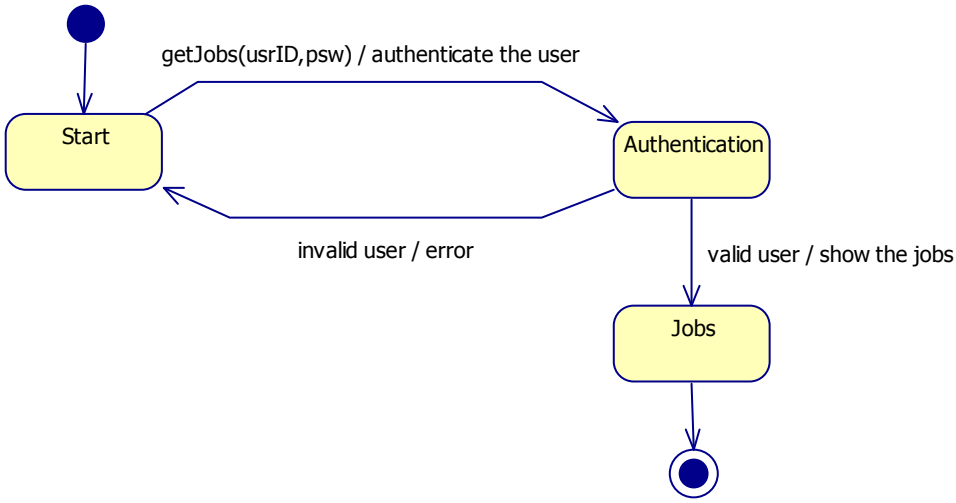


Figure 31. States Diagram: Get Jobs

V. Is Available

From the initial state, when the system receives an isAvailable request, the availability is checked and is returned.

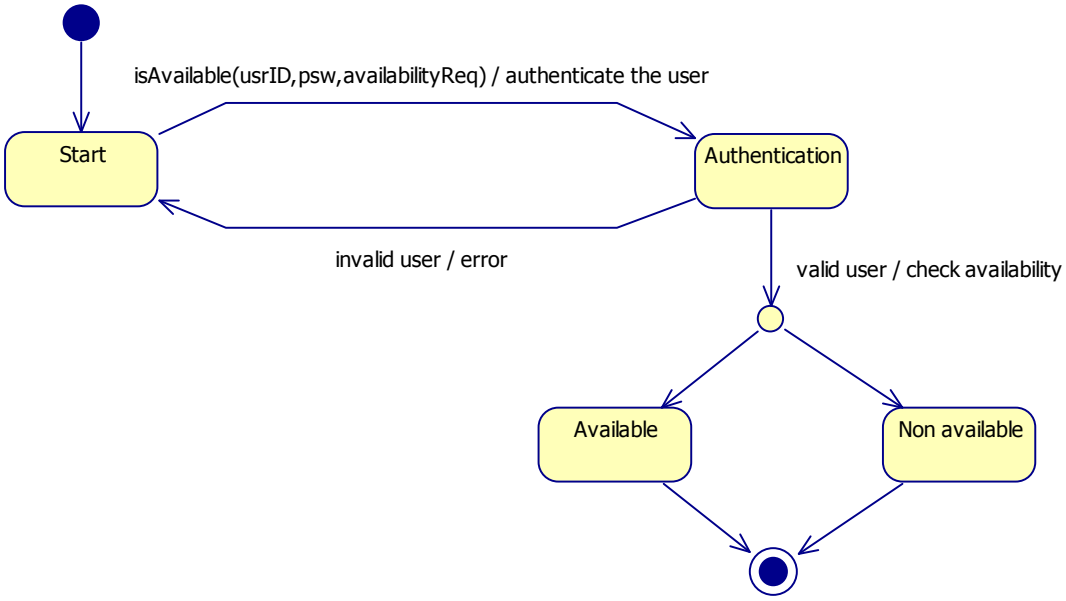


Figure 32. States Diagram: Is Available

VI. Create Reservation

It can be seen how from the initial state, when the system receives a createReservation request, the system checks the availability, registers the reservation, schedules it and returns a message to the user indicating if the creation of the reservation has been successful.

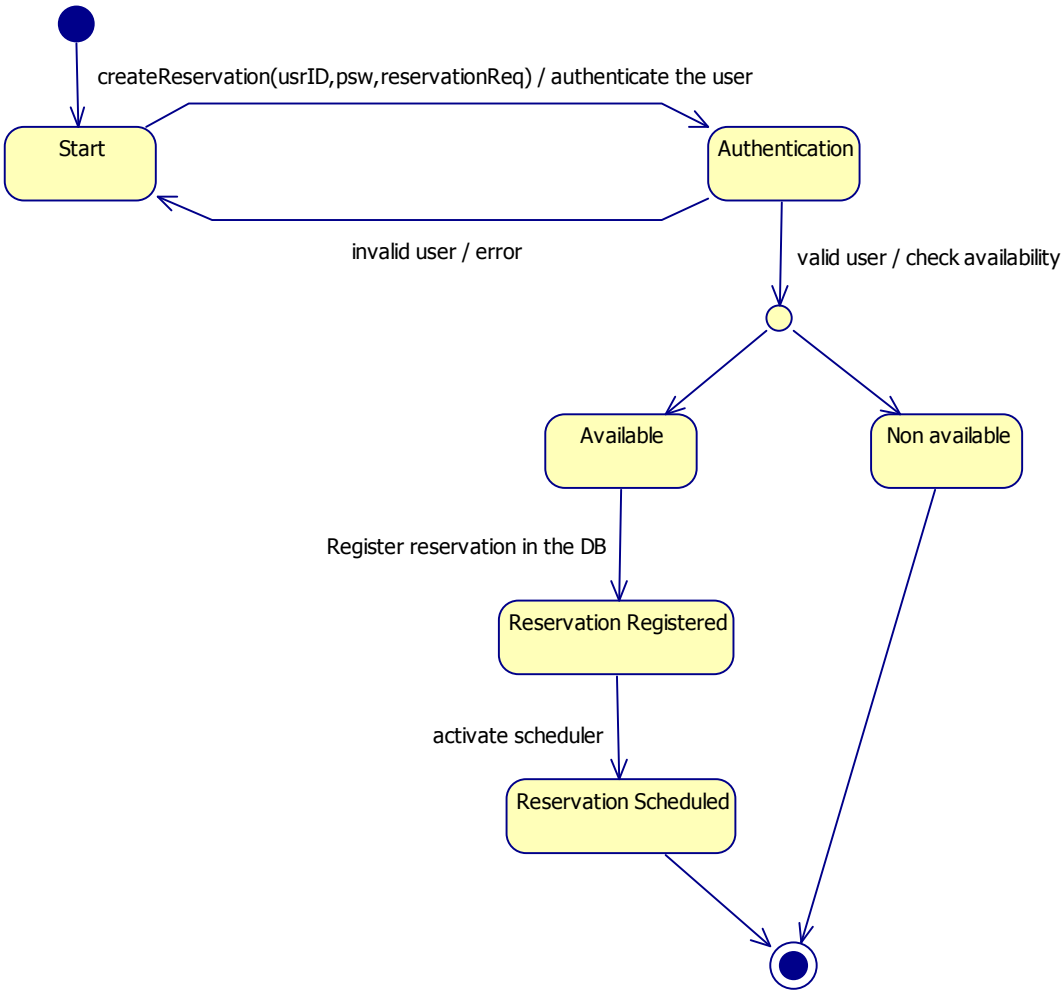


Figure 33. States Diagram: Create Reservation

VII. Cancel Reservation

When the system receives a cancelReservation request, if the reservation exists, the system makes de corresponding deletions in the database and returns a message to the user informing about the success of the operation.

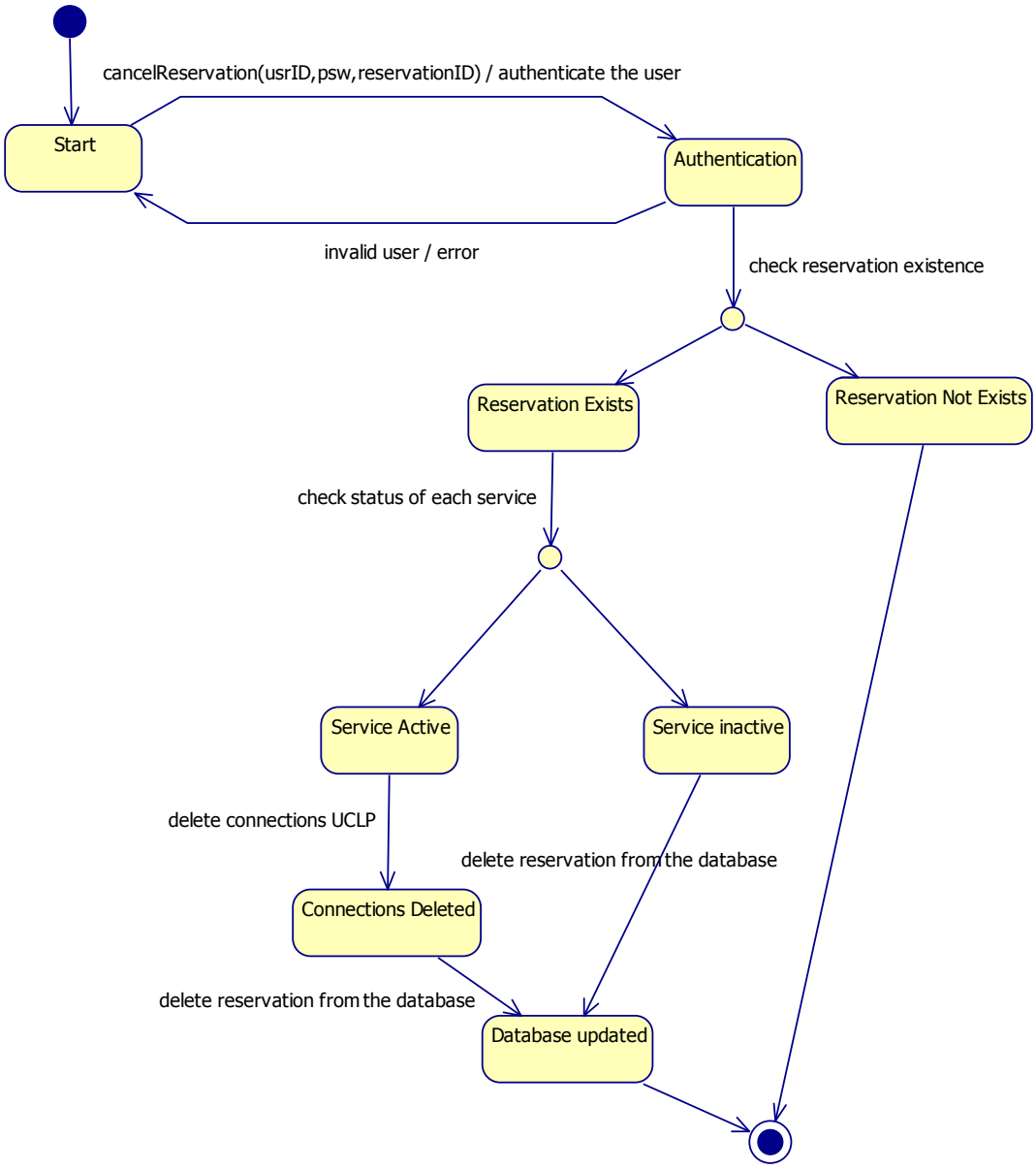


Figure 34. States Diagram: Cancel Reservation

VIII. Get Status

If the reservation requested exists, the system shows to the user the status of each service of the reservation.

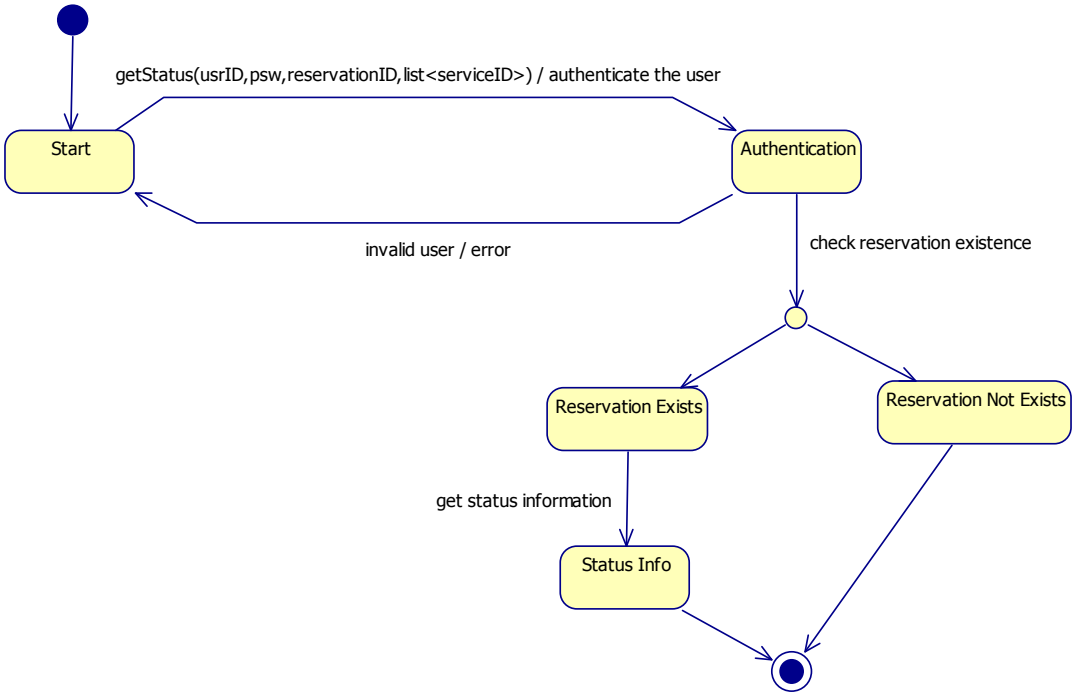


Figure 35. States Diagram: Get Status

IX. Complete Job

The system converts all the pre-reservations of the job into permanent reservations and schedules them. The system shows a message to the user indicating if the operation has been successful.

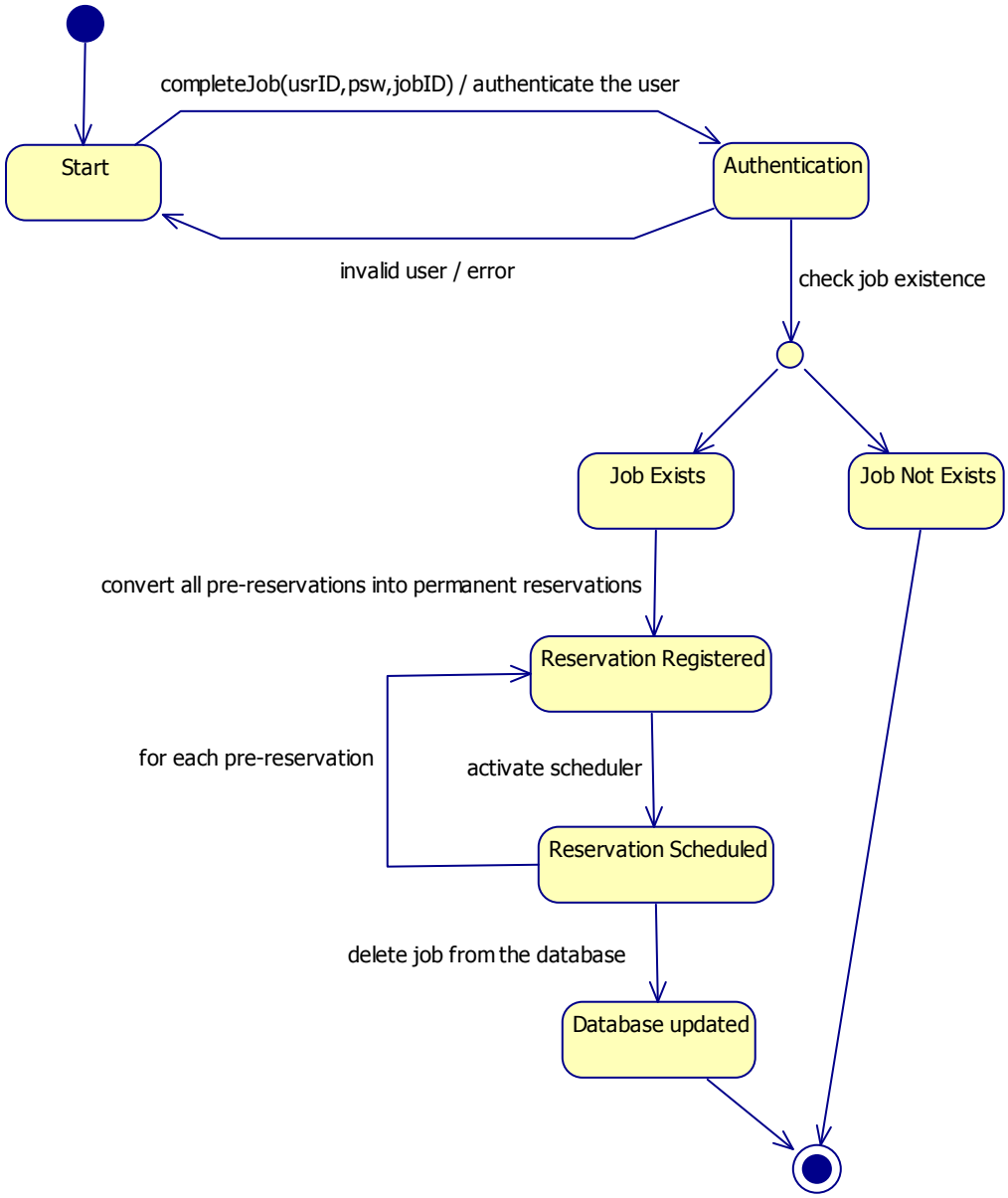


Figure 36. States Diagram: Complete Job

X. Cancel Job

If the job exists, it is deleted from the database and also its pre-reservations.

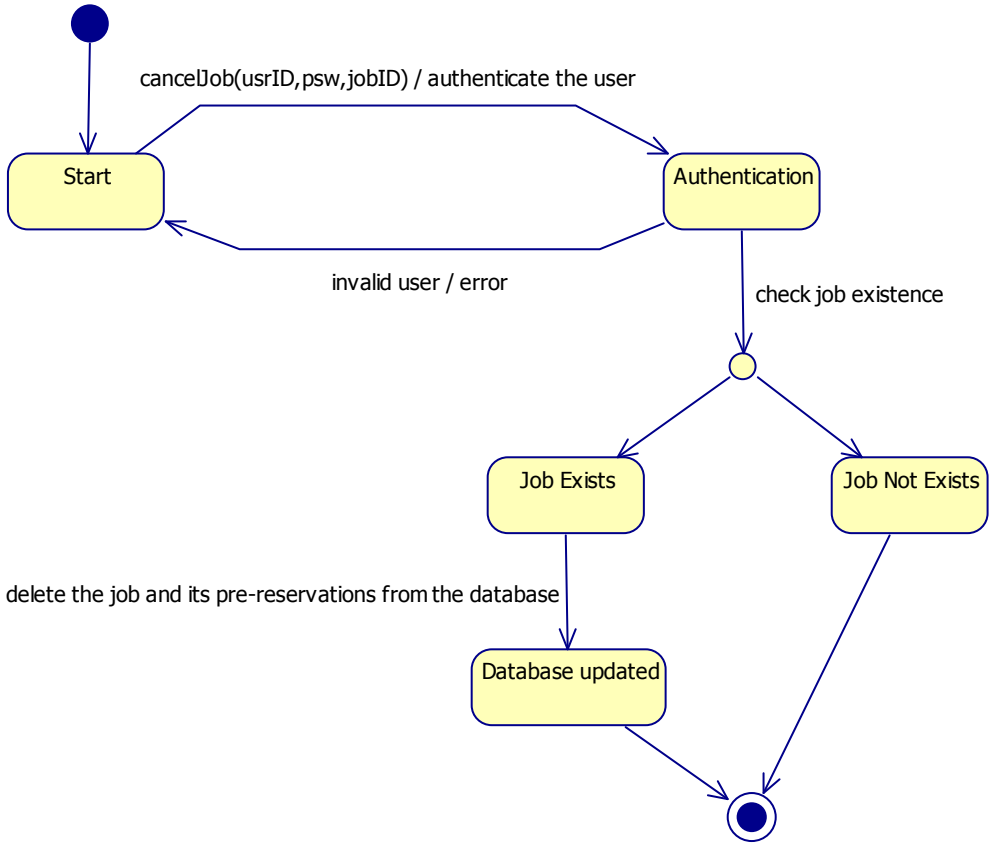


Figure 37. States Diagram: Cancel Job

XI. Activate

It can be seen how from the initial state, the existence of the service is checked and if it is successful the connections of the service are activated.

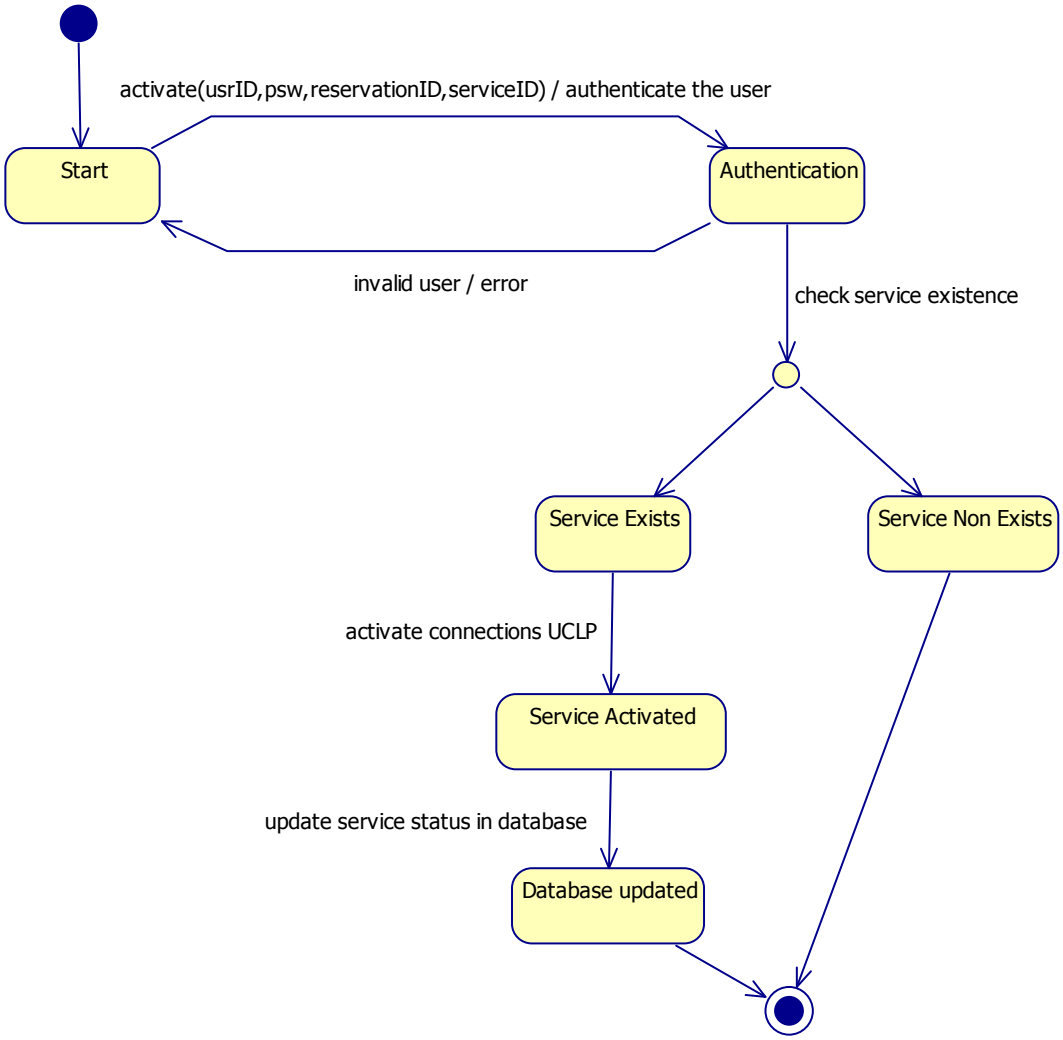


Figure 38. States Diagram: Activate

XII. Activate Automatically

The connections of the service are activated and the database updated.

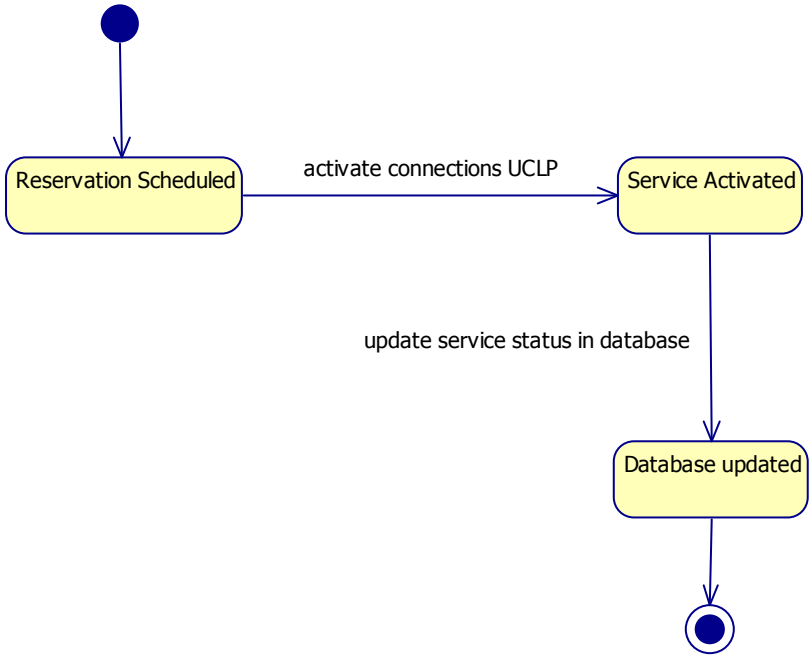


Figure 39. States Diagram: Activate Automatically

XIII. Complete Service

When a service is active and it has to be completed the system deletes all the connections of the service and updates the database.

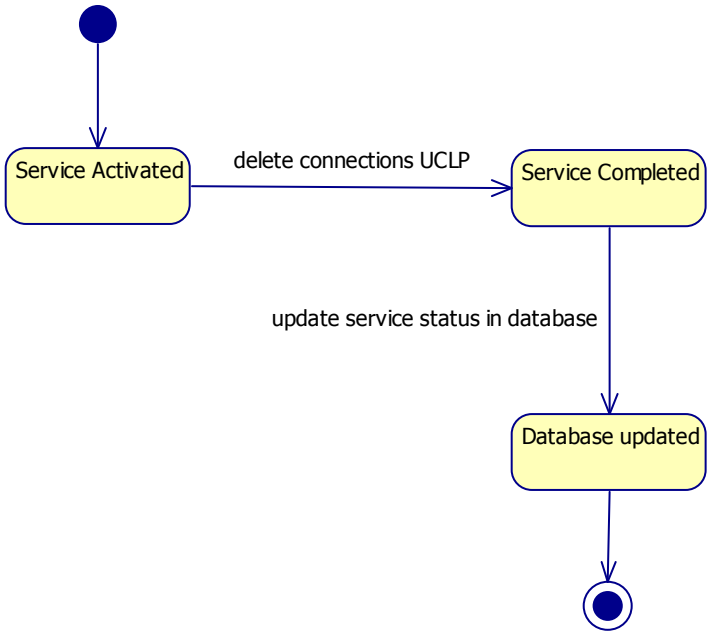


Figure 40. States Diagram: Complete Service

XIV. Reconfigure System

When the service is started up, the system checks which connection must be reconfigured.

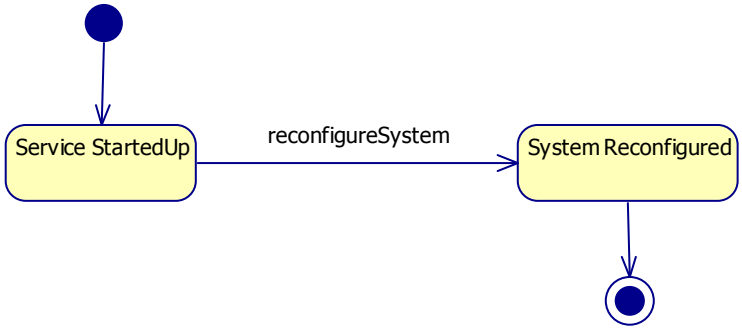


Figure 41. States Diagram: Reconfigure System

8. DESIGN

The object oriented design is a refinement of the model obtained in the specification process. What is wanted is to model the intern architecture of the system and identify its components.

8.1. INTRODUCTION

Once the analysis of requirements and the specification of the software are done and are clear, the different options that the Software Engineering offers must be chosen in order to get an application as satisfactory as possible. It must be considered that the decisions made are not absolutely; that is, a decision never is definitive, so it is necessary to reach en equilibrium between the advantages and the disadvantages that are involved in the decisions.

8.1.1. Logical Architecture of the system

The starting point will be a logical architecture of an information system in different layers, as seen in the figure number 41.

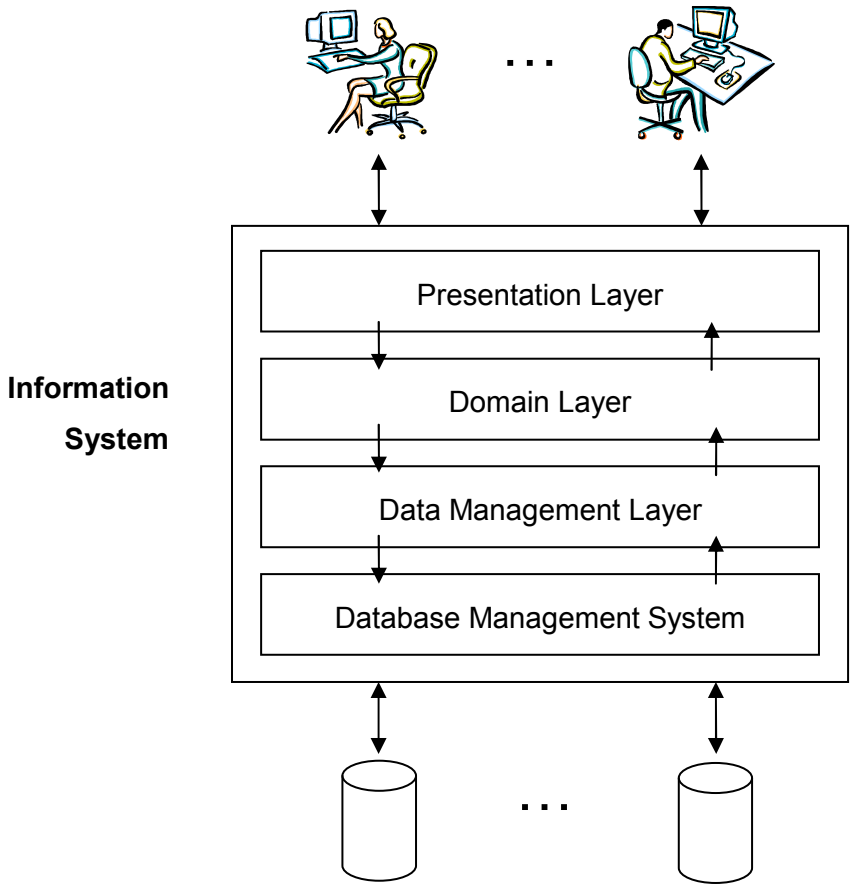


Figure 42. Generic Information System divided into layers

Presentation layer: Orders the executions of actions, gathers events and manages the interface. It is the responsible for the interaction between the user and the system.

Domain layer: Executes the requested actions, obtains the results and communicates the responses to the presentation layer. It is the responsible for the implementation of the functionalities of the system.

Data Management layer: It provides independence to the data in order to facilitate the domain to treat the information without worrying about where is the data stored. It is the responsible for the interaction between the domain and the database.

Database Management System: Makes the requested queries to the database. It is the responsible for the correct and persistent storage and the maintenance of the data.

As explained in section 4.2, the actual users of the ARS are the ones from the PHOSPHORUS Consortium, and so at the moment this project does not contain a presentation layer. At present, the project is designed without this layer because the actual users don't need a graphical interface to interact with the system; they interact with the ARS directly through an interface of the domain layer.

Actually, a simple web application is described in section 10 (Testing). This application allows a human user to interact with the ARS directly through a GUI, but this GUI has been only developed in order to test the ARS. In the future, this web application could be developed in deep in order to allow other human users to interact with the ARS if necessary and then, this GUI will be the presentation layer of the project.

Thus, as at present no presentation layer is needed, it is not described. So, the layer distribution of this system will be like the following picture.

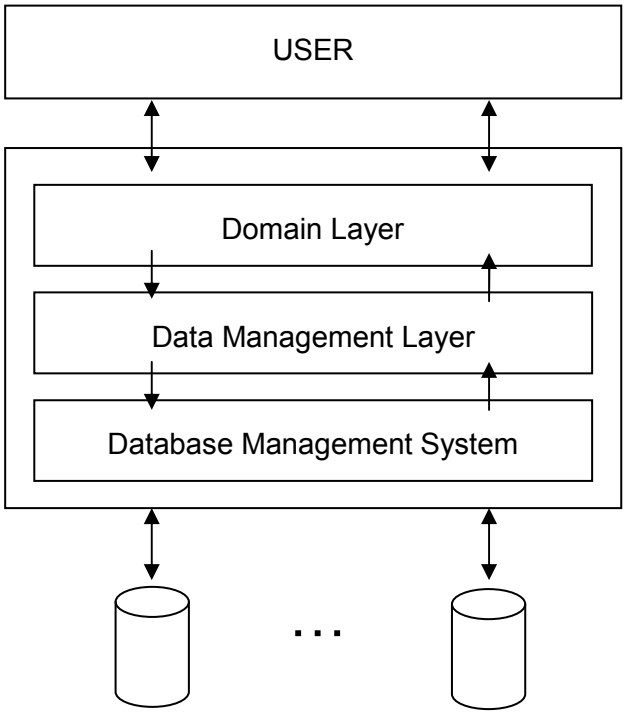


Figure 43. Logical Architecture of the ARS

Domain layer: In this case, this layer communicates directly with the PHOSPHORUS interface.

Data Management layer: It is the responsible for the interaction between the domain and the database.

Database Management System: Makes the requested queries to the database.

Therefore, this section will be divided in two subsections:

- design of the domain layer
- design of the data management layer

8.2. Design of the Domain Layer

This section contains the sequence diagrams of the domain layer corresponding to the use cases specified in the section 4.4 (The Scope of the Work). These sequence diagrams shows the interaction between the different elements of the system to get a response for request done by the user.

In the following diagrams, there are two entities that are present in all diagrams. These ones are the responsible for the communication with the users of the system and for the execution of the actions. These entities are the WS and the Controller.

The WS (Web Service) is the responsible for the communication with the users. The WS is implemented as an interface that receives the requests, transforms the information in terms of the Controller can understand it, and gives it to him, who processes the request, makes the computations and gives the responses to the WS. This one transforms the information again and returns it to the end users. As it has been mentioned before, this process is done by means of an interface, which will be explained in deep in the Implementation section (the next one, number 9).

When the system is started up, the WS is also the one that reads the topology file from the UCLP system and indicates the Controller which endpoints, nodes and links the network has; thus, the Controller creates the network structure. WS is also the responsible for the authentication of the user. To sum up, the WS is the responsible for executing the following actions:

- Retrieve the topology file from the UCLP system.
- Authenticate the user.
- Receive the request, process the information and call the Controller to perform the requested functionality.
- Return a response to the user with the correct format of the data, by means of transforming the information returned by the Controller in terms of information understandable for the user.

For each sequence diagram, there is a little explanation about the work of the operation, the input and output parameters that the WS receives and returns to the user and the cost of the operation. Some diagrams are divided in more diagrams; that is, in order to the better comprehension of the sequence diagrams, as these ones are too large, some part of some diagrams has been done in another diagram; this diagram has the same name as the function that the first diagram calls (it would be like follow a link).

8.2.1. Get Features

In the figure 42 is shown the sequence diagram for the use case ‘Get Features’. The WS receives the request from the user to return the features of the system, it contacts the controller, who returns the list of the features to the WS, and this one returns this list to the user.

Input: GetFeatures

- AAAUserType
    - String userID

- String password

Output: GetFeaturesResponse

- String[] features

Cost: Θ(1)

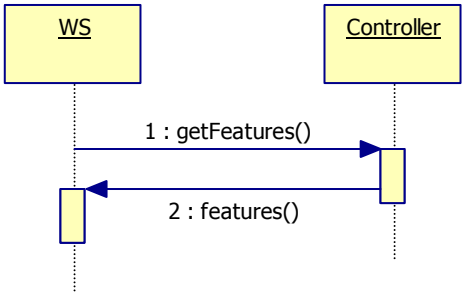


Figure 44. Get Features

8.2.2. Get Endpoints

In the next figure is shown the sequence diagram for the use case 'Get Endpoints'. The WS receives the request from the user to return the available endpoints of the network; it contacts the controller, who gets the endpoints from the net, creates the list to be returned and returns it to the WS, who returns the list to the user.

Input: GetEndpoints

- AAAUserType
  - String userID
  - String password

Output: GetEndpointsResponse

- EndpointType[] endpoints
  - String endpointID
  - String name
  - String description
  - EndpointInterfaceType interface
  - String domainID
  - Int bandwidth

Cost: Θ(E)

E = number of endpoints

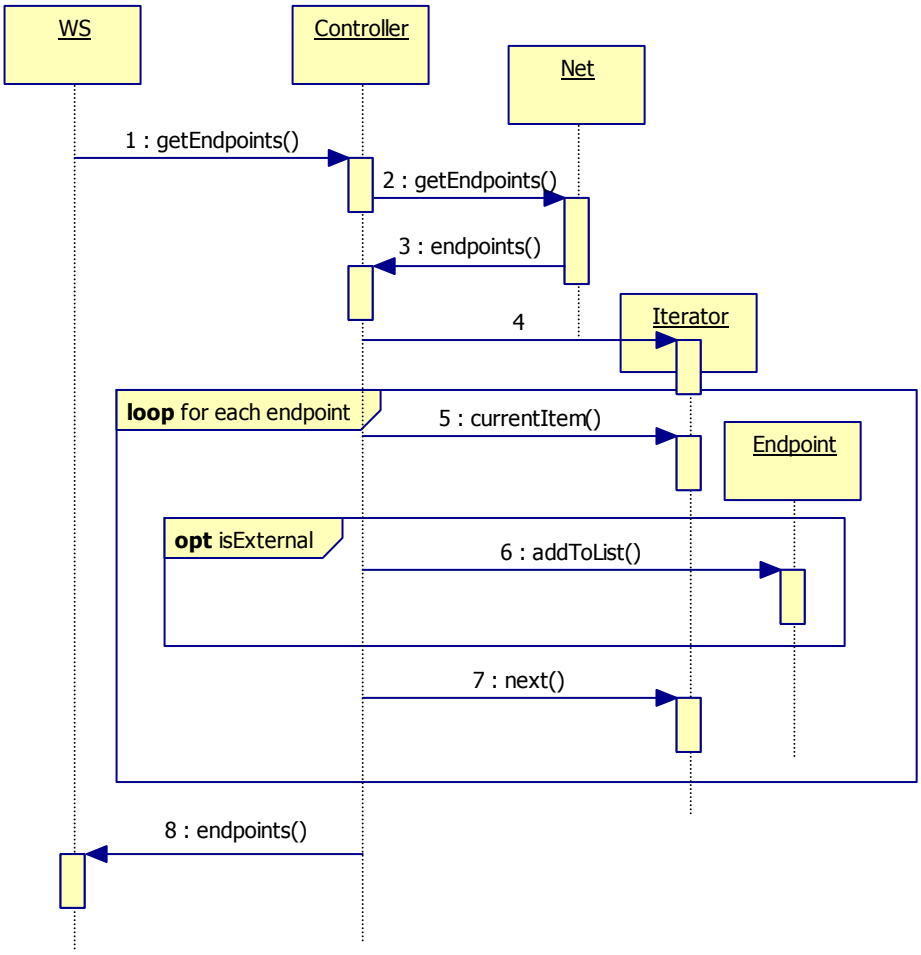


Figure 45. Get Endpoints

8.2.3. Get Reservations

In the figure below is shown the sequence diagram for the use case 'Get Reservations'. The WS receives the request from the user; it contacts the controller, who queries to the database the reservations done, and returns them to the WS.

Input: GetReservations

- AAAUserType
  - String userID
  - String password

Output: GetReservationsResponse

- ReservationInfo[] reservations
  - Long reservationID
  - Boolean isPre
  - ServiceInfo[] services
    - int serviced
    - Boolean automaticActivation
    - String status

- ReservationType typeOfReservation
- FixedReservationContraintType fixed
- DeferrableReservationContraintType deferrable
- MalleableReservationContraintType malleable
- ConnectionInfo[] connections
    - Int connectionID
    - String status
    - EndpointInfo[] endpoints
        - Int endpointID
        - String interfaceType

Cost: Θ(1)

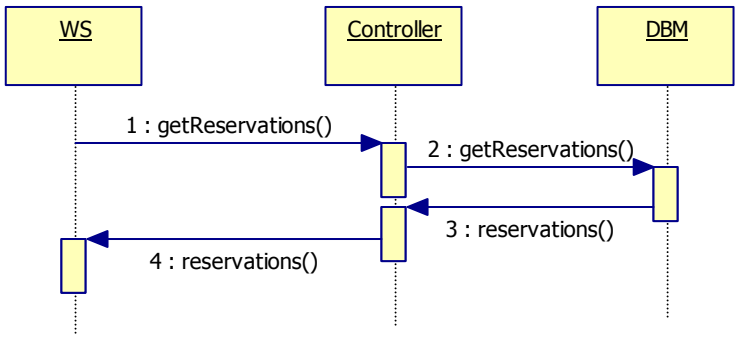


Figure 46. Get Reservations

8.2.4. Get Jobs

In the figure below is shown the sequence diagram for the use case 'Get Jobs. The WS receives the request from the user; it contacts the controller, who queries to the database the jobs done, and returns them to the WS.

Input: GetJobs

- AAAUserType
    - String userID
    - String password

Output: GetJobsResponse

- JobInfo[] jobs
    - Long jobID
    - ReservationInfo[] pre-reservations
        - Long reservationID
        - Boolean isPre
        - ServiceInfo[] services
            - int serviced
            - Boolean automaticActivation

- String status
- ReservationType typeOfReservation
- FixedReservationContraintType fixed
- DeferrableReservationContraintType deferrable
- MalleableReservationContraintType malleable
- ConnectionInfo[] connections
    - Int connectionID
    - String status
    - EndpointInfo[] endpoints
        - Int endpointID
        - String interfaceType

Cost: Θ(1)

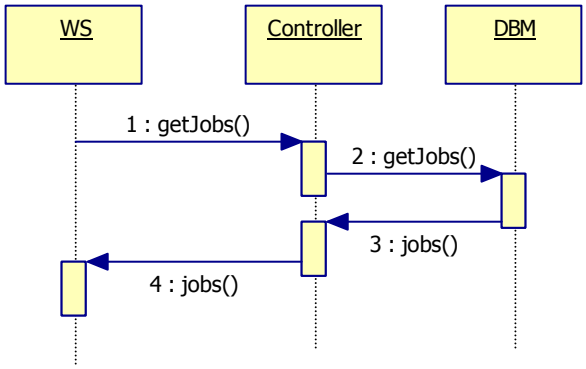


Figure 47. Get Jobs

8.2.5. Is Available

In the next figure is shown the sequence diagram for the use case 'is Available'. The WS receives the request from the user and it contacts the controller. The controller creates a reservation and for each service requested checks that the ID requested is valid and, if it is valid, creates a new service, sets its attributes and associates the reservation to this service. For each connection requested of the service, the controller checks if the ID requested is valid; if it is valid it creates a new connection and sets its attributes, associates the connection to the service, and calculates if a path exists for the source and target endpoints requested. If a possible path exists, the controller creates a new ConnectionEndpoint for each endpoint of the path, associates these ConnectionEndpoint's to the connection and marks the connection as available. Finally, if there is some connection requested not available, an alternative start time is calculated and a response indicating that the request is not available is returned. Otherwise, an affirmative response is returned to the user.

Input: IsAvailable

- ServiceConstraintType[] service
    - int serviceID
    - ReservationType typeOfReservation
    - FixedReservationConstraintType (optional)
        - Calendar startTime
        - Int duration
    - DeferrableReservationConstraintType (optional)
        - Calendar earliestStartTime
        - Int duration
        - Calendar deadline
    - MalleableReservationContraintType (optional)
        - Calendar earliestStartTime
        - Calendar deadline
    - Boolean automaticActivation
    - ConnectionConstraintType[] connections
        - int connectionID
        - EndpointType source
        - EndpointType target
        - Int directionality
        - int minBW
        - int maxBW
        - Int maxDelay
        - Long dataAmount
- Long jobID
- AAAUserType
    - Sting userID

- String password

Output: IsAvailableResponse

- ConnectionAvailabilityType [] detailedResult
  - int serviceID
  - int connectionID
  - Int availability
- Long alternativeStartTimeOffset (optional)

Cost: O(S * vi * C * vi * cp + calt)

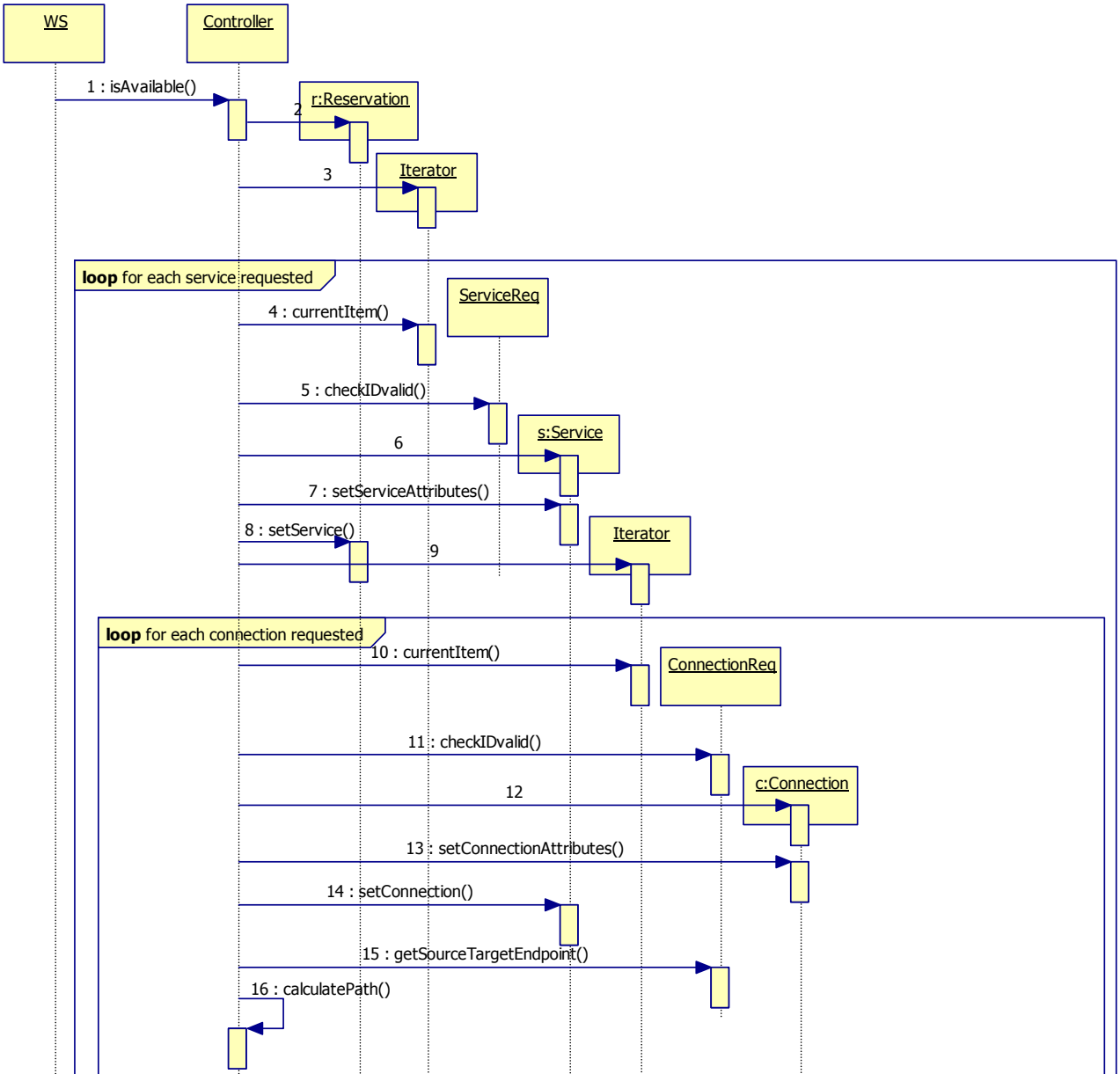
S = # services requested

vi = cost compute validEndpoint

C = # connections requested

cp = cost compute calculatePath

calt = cost compute calculateAlternativeStartTime



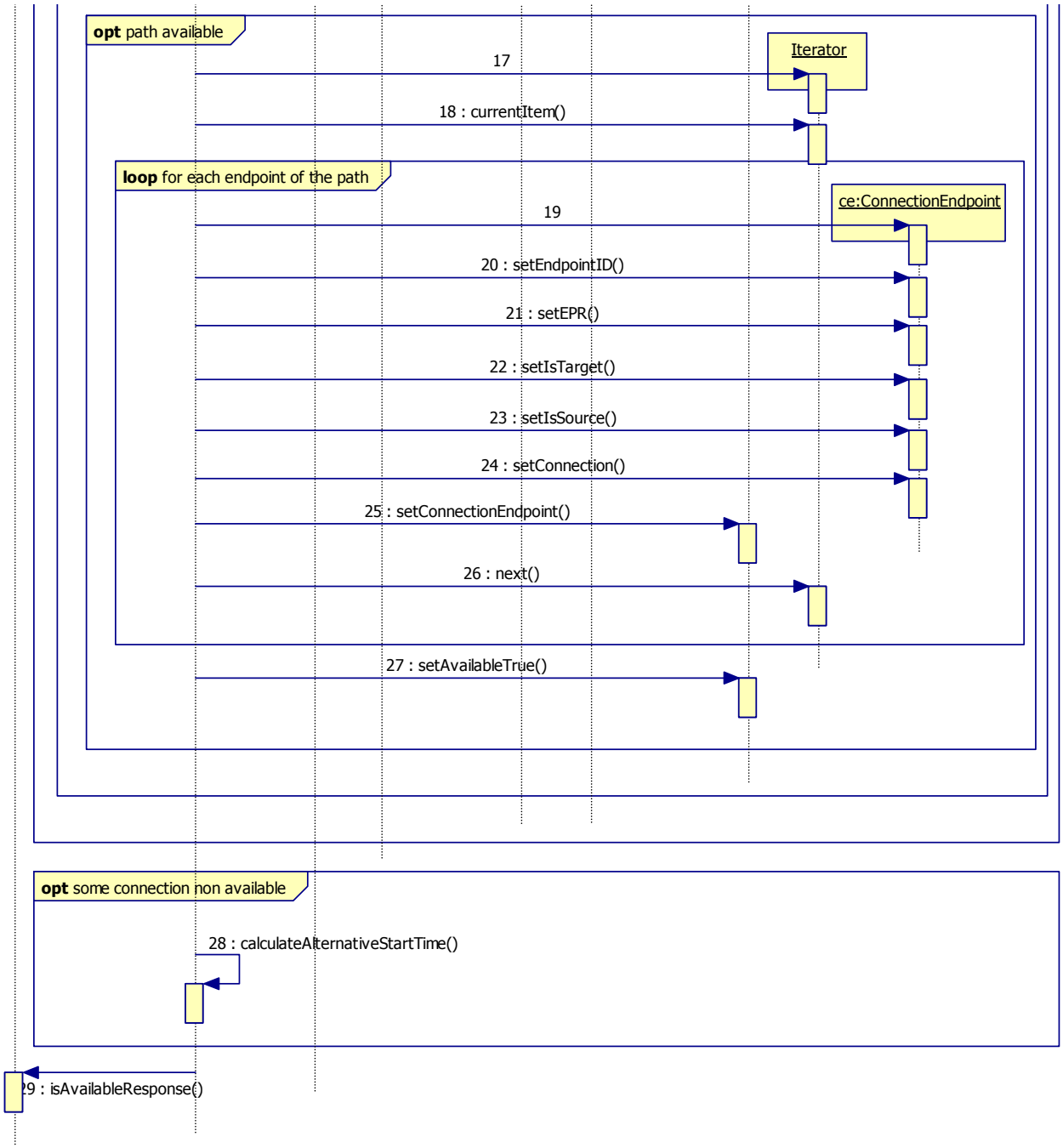


Figure 48. Is Available

CheckIDvalid

The actual ID is compared with the other identifiers of the elements of the list. If some ID is repeated, the identifier is not valid, so FALSE is returned.

Cost: Θ(IT)

IT = # items

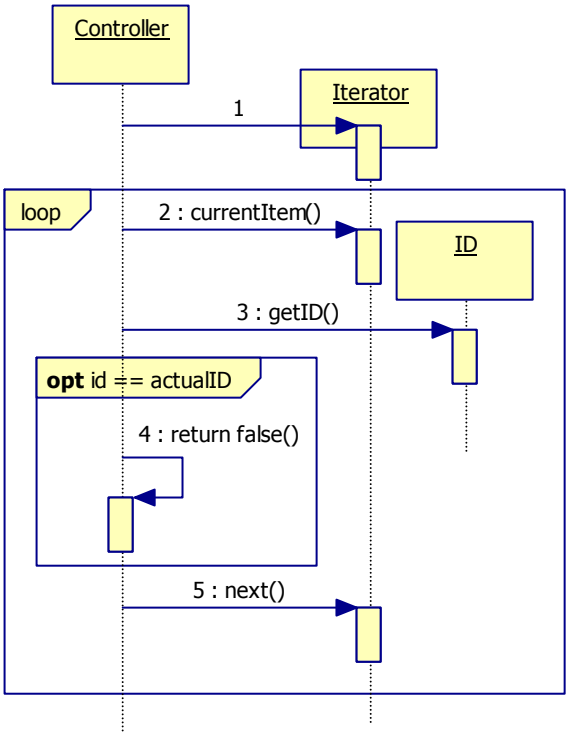


Figure 49. Check ID valid

setServiceAttributes

The controller sets to the service the following attributes: the identifier, the automatic activation, the status as 'pending', the type of the reservation and depending on this one sets the fixed reservation constraints, or the deferrable or the malleable ones. And finally associates the reservation to the service.

Cost: Θ(1)

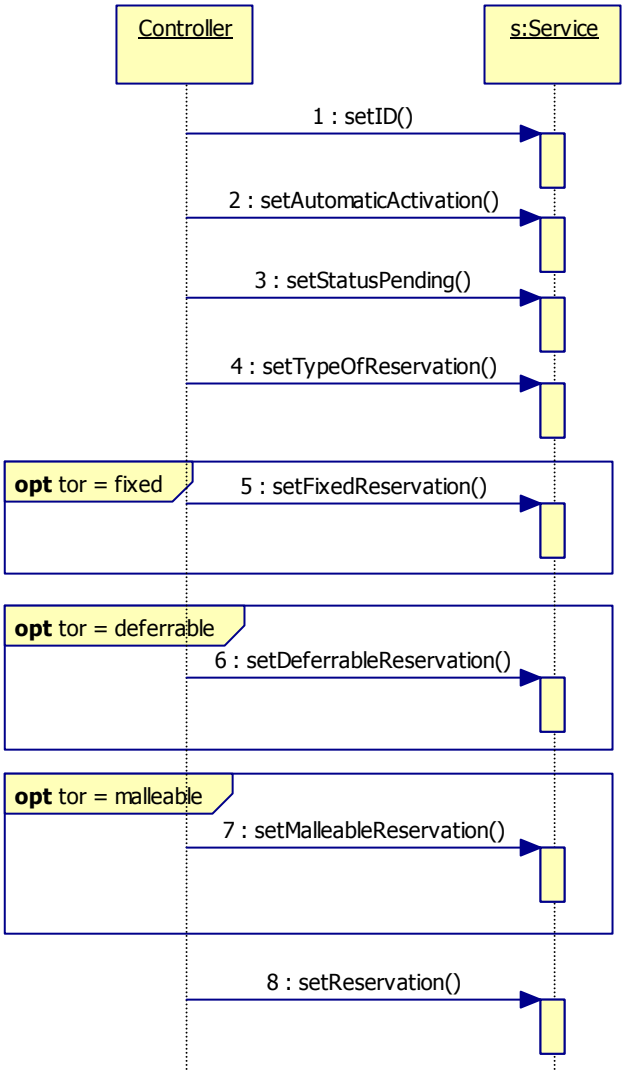


Figure 50. Set Service Attributes

setConnectionsAttributes

The controller sets to the connection the following attributes: the identifier, the directionality, the maximum bandwidth, the minimum bandwidth, the actual bandwidth, the delay and the status as 'pending'. Finally associates the service to the connection.

Cost: Θ(1)

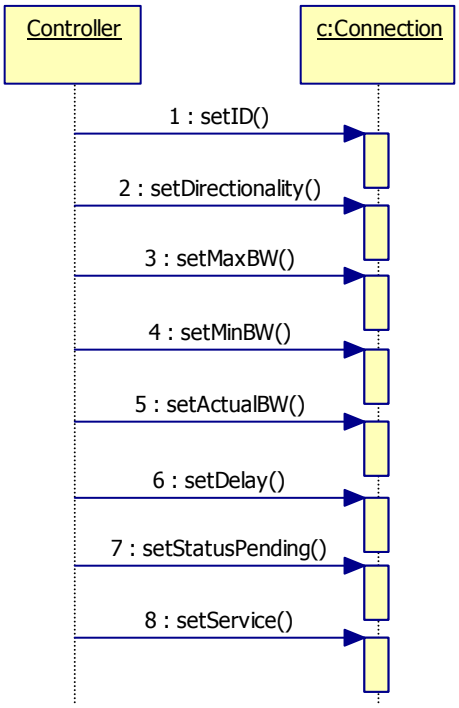


Figure 51. Set Connections Attributes

calculatePath

If the source and the target endpoints of the connection are valid, for each link of the network is checked if this link is available; if the link is not available it is removed from the list of links. Finally, with the list of the links available, the Dijsktra algorithm is computed. The output of the algorithm, if a path is available, is a list of the endpoints of the route, otherwise a negative response is returned.

Cost: Θ(2ve * L * la * DSP)

ve = cost compute validEndpoint

L = # links

la = cost compute linkAvailable

DSP = cost compute the DSP algorithm

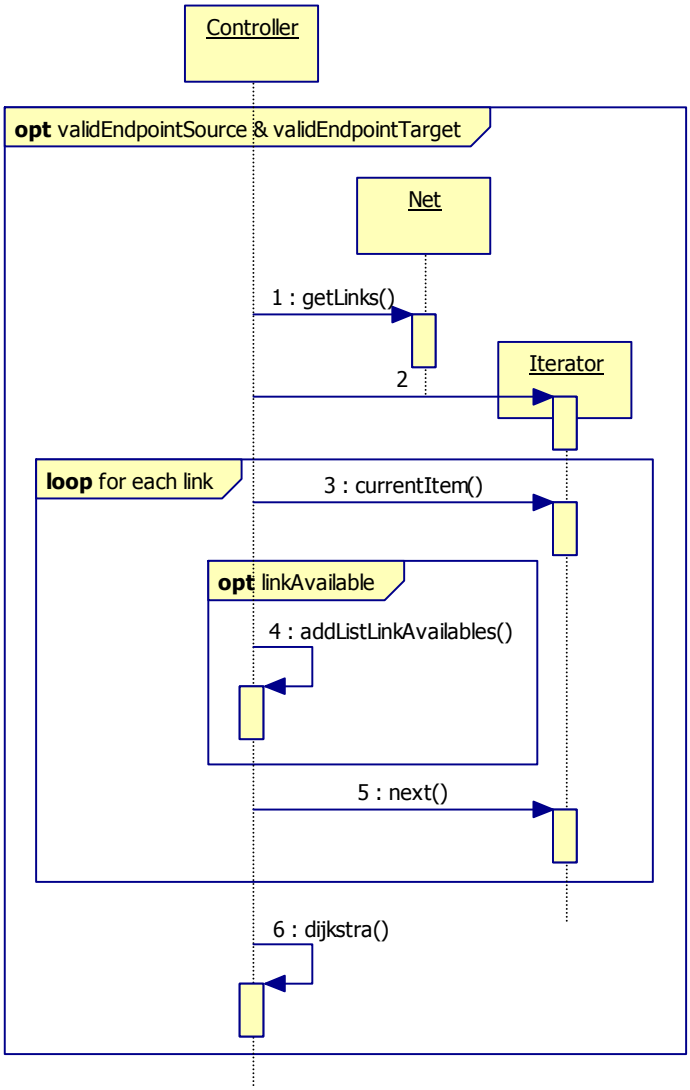


Figure 52. Calculate Path

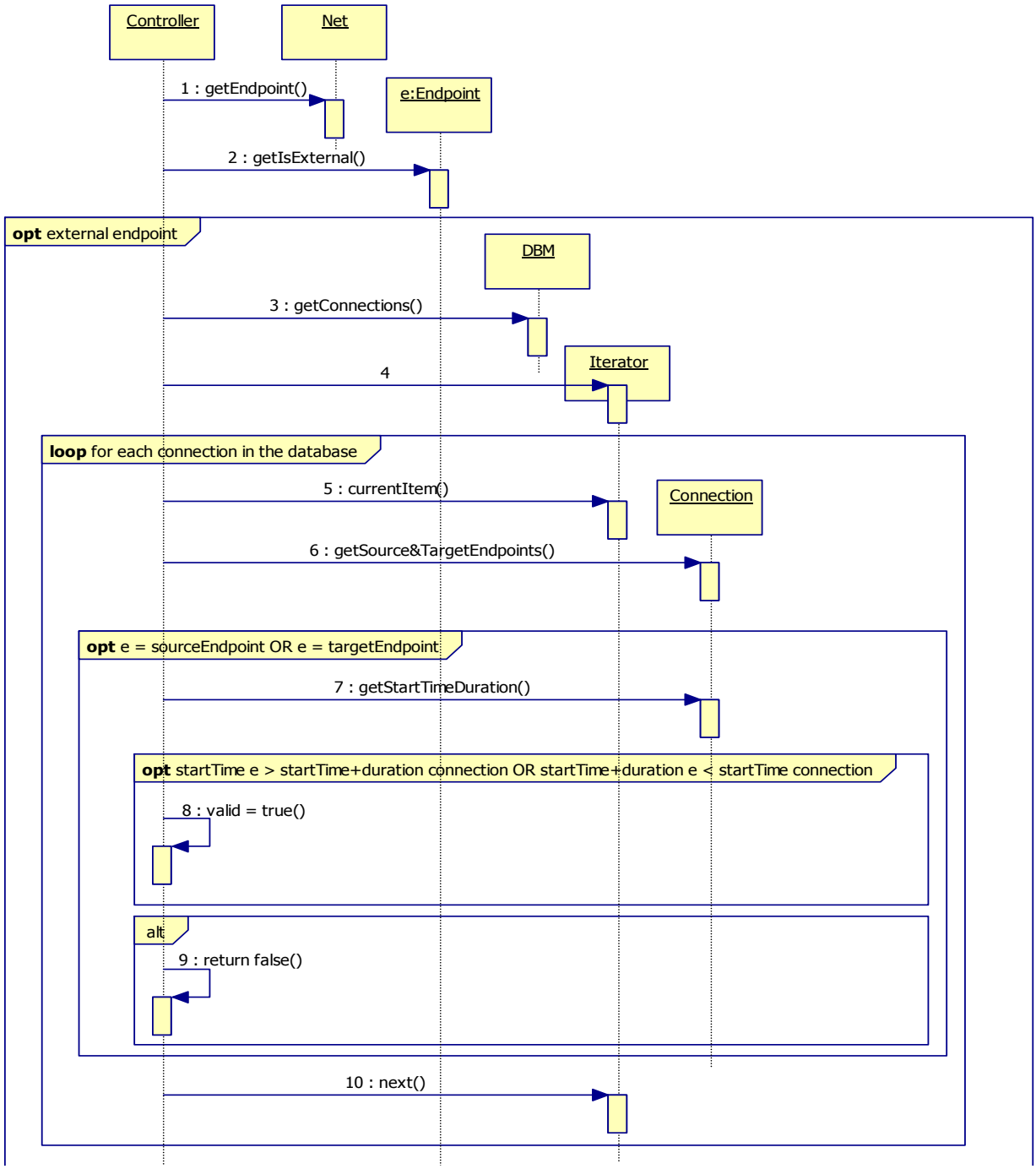
validEndpoint

The controller gets the endpoint of the net that has the same EPR that the one requested. If this endpoint is not external, is not valid to create a connection. If it is valid, the controller queries to the database all the connections that contain this endpoint in its path. For each endpoint of each connection, if the endpoint is the same than the one that is being checked, the start time and duration of both are compared. If the start time of the endpoint being checked is smaller than the start time plus the duration of the connection, and the start time plus the duration of the endpoint being checked is bigger than the start time of the connection, then the endpoint is not valid. If the endpoint requested is compatible with each connection, it must be check its compatibility with the other connections of the reservation requested, and it is done like it has been done with the existing connections in the database.

Cost: Θ(CDB * CR)

CDB = # connections in the database

CR = # connections requested


Controller
Net
e:Endpoint
1 : getEndpoint()
2 : getIsExternal()
opt external endpoint
DBM
3 : getConnections()
Iterator
4
loop for each connection in the database
5 : currentItem()
Connection
6 : getSource&TargetEndpoints()
opt e = sourceEndpoint OR e = targetEndpoint
7 : getStartTimeDuration()
opt startTime e > startTime+duration connection OR startTime+duration e < startTime connection
8 : valid = true()
alt
9 : return false()
10 : next()

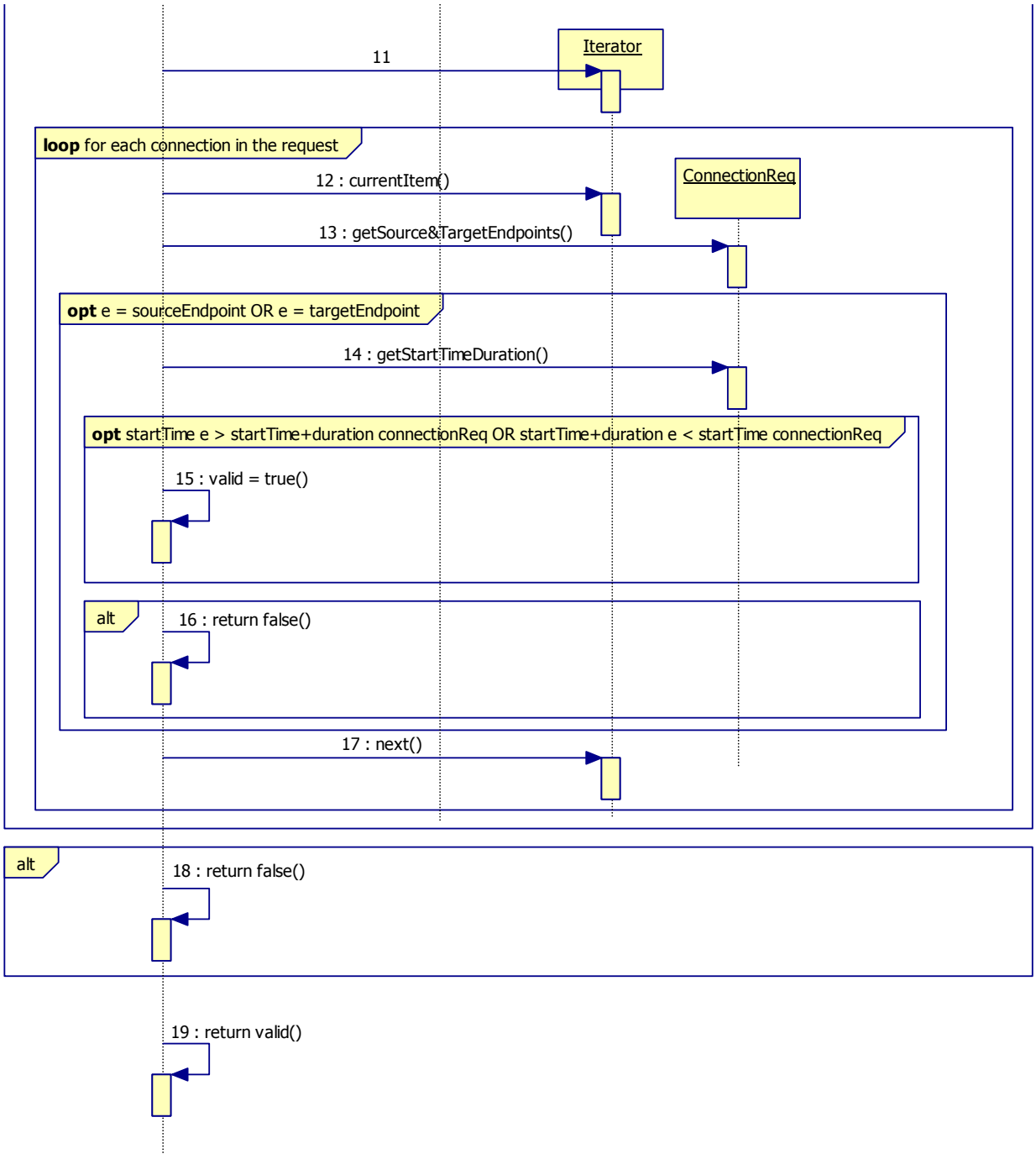


Figure 53. Valid Endpoint

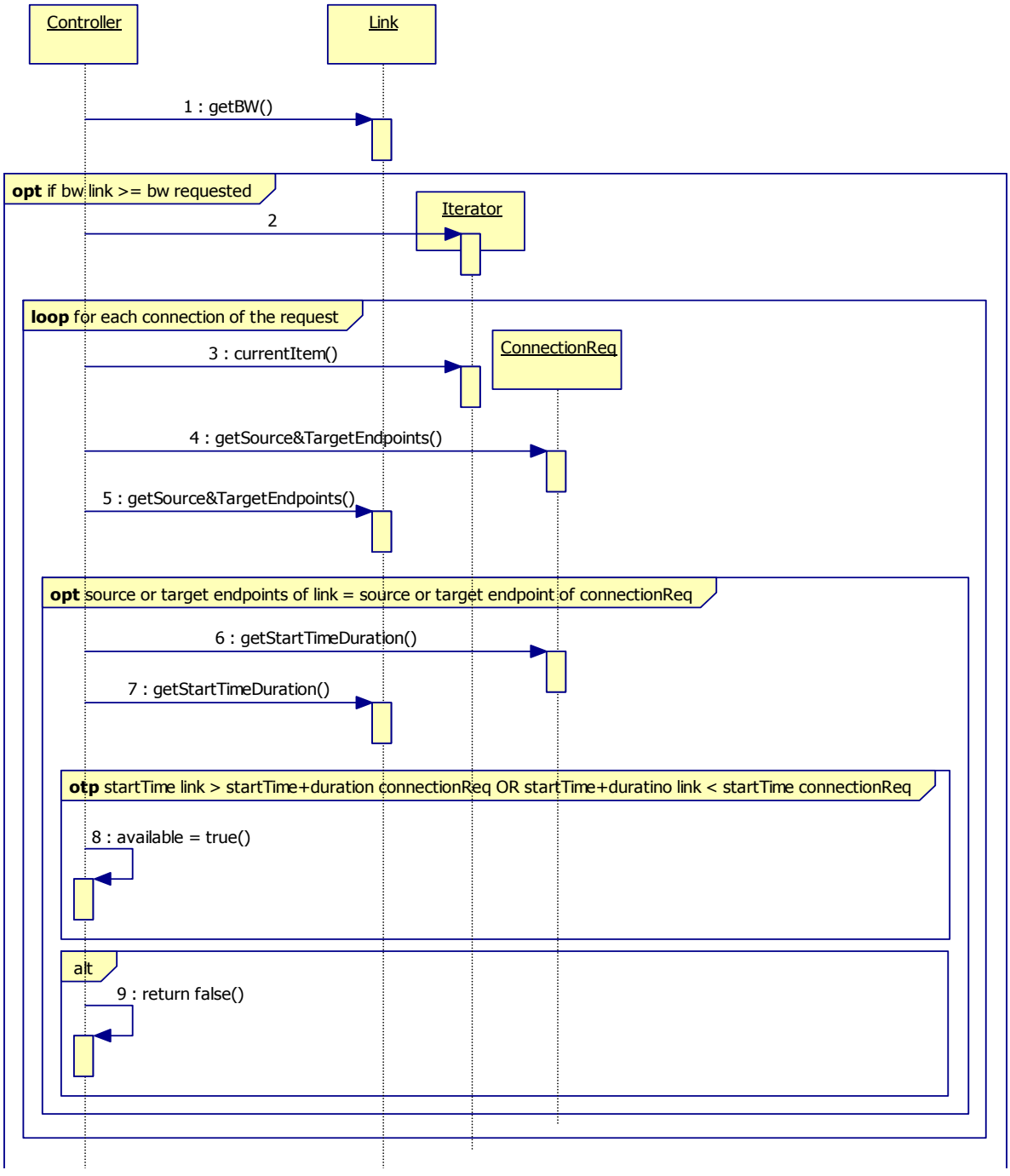
linkAvailable

First of all, the bandwidth of the link is checked; if the bandwidth of the link is smaller than the requested one, the link is not available. For each connection of the reservation requested, the start time and duration of endpoints of this one are compared with the start time and duration of the link in the same way as it is done in the last operation. If the times are not compatible, the link is not available. Otherwise the start time and duration of the link are compared with the start time and duration of the connections existing in the database. If the link is not compatible with some connection, it is not valid; otherwise, the link is valid.

Cost: Θ(CDB * CR)

CDB = # connections in the database

CR = # connections requested



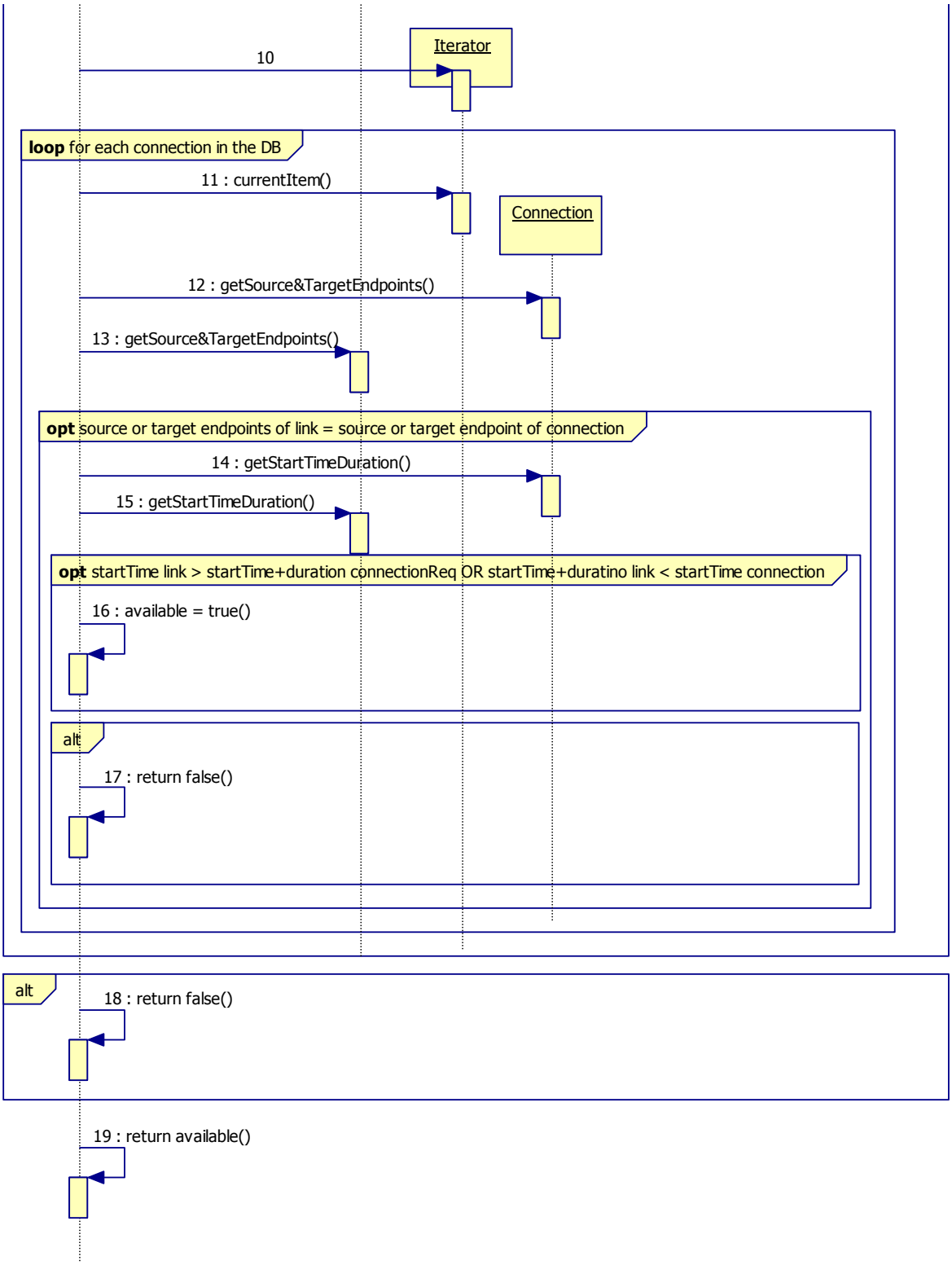


Figure 54. Link available

calculateAlternativeStartTime

The availability of the reservation requested is checked with different start time's until one possible start time is valid or until a maximum start time is reached. If exists a possible start time, this one is returned; otherwise no alternative start time is returned. In each loop, the possible start time is incremented.

Cost: O(cp * MAX)

cp = cost compute calculatePath

MAX = number maximum of iterations

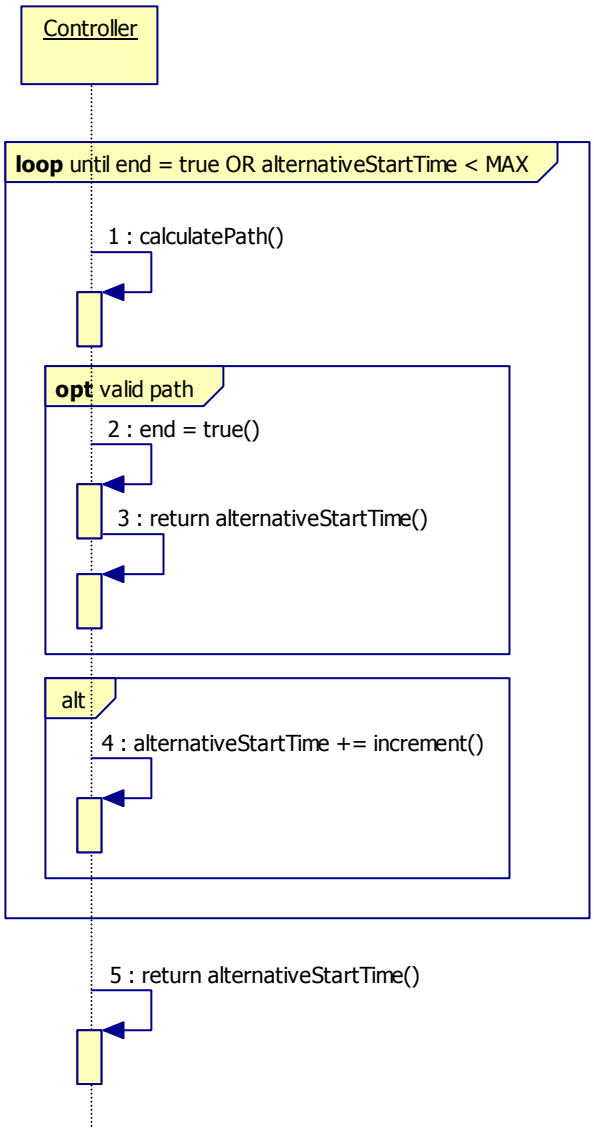


Figure 55. Calculate alternative start time

8.2.6. Create Reservation

In the figure number 54 is shown the sequence diagram for the use case ‘create Reservation’. The steps carried out are the same than in the use case ‘is Available’. But in this case, if the reservation is possible, it is stored in the database. Then the jobID given in the request is checked. If it is null, a permanent reservation is requested and each service is programmed with the scheduler to be activated at the start time requested. If the jobID is equal to zero, a pre-reservation is requested and a new job has to be created; the job and the pre-reservation are associated and stored in the database. Otherwise, if the jobID is larger than zero, a pre-reservation is requested and this one must be associated to an existent job; the existence of the job is checked and if it exists, this one and the pre-reservation are associated and stored in the database. If the reservation requested is not possible, or the jobID given does not exist, an exception is thrown and the reservation is not created.

Input: CreateReservation

- ServiceConstraintType[] service
    - int serviceID
    - ReservationType typeOfReservation
    - FixedReservationConstraintType (optional)
        - Calendar startTime
        - Int duration
    - DeferrableReservationConstraintType (optional)
        - Calendar earliestStartTime
        - Int duration
        - Calendar deadline
    - MalleableReservationContraintType (optional)
        - Calendar earliestStartTime
        - Calendar deadline
    - Boolean automaticActivation
    - ConnectionConstraintType[] connections
        - int connectionID
        - EndpointType source
        - EndpointType target
        - Int directionality
        - int minBW
        - int maxBW
        - Int maxDelay
        - Long dataAmount
- Long jobID
- String notificationConsumerURL
- AAAUserType

- Sting userID
- String password

Output: CreateReservationResponse

- Long reservationID
- Long jobID

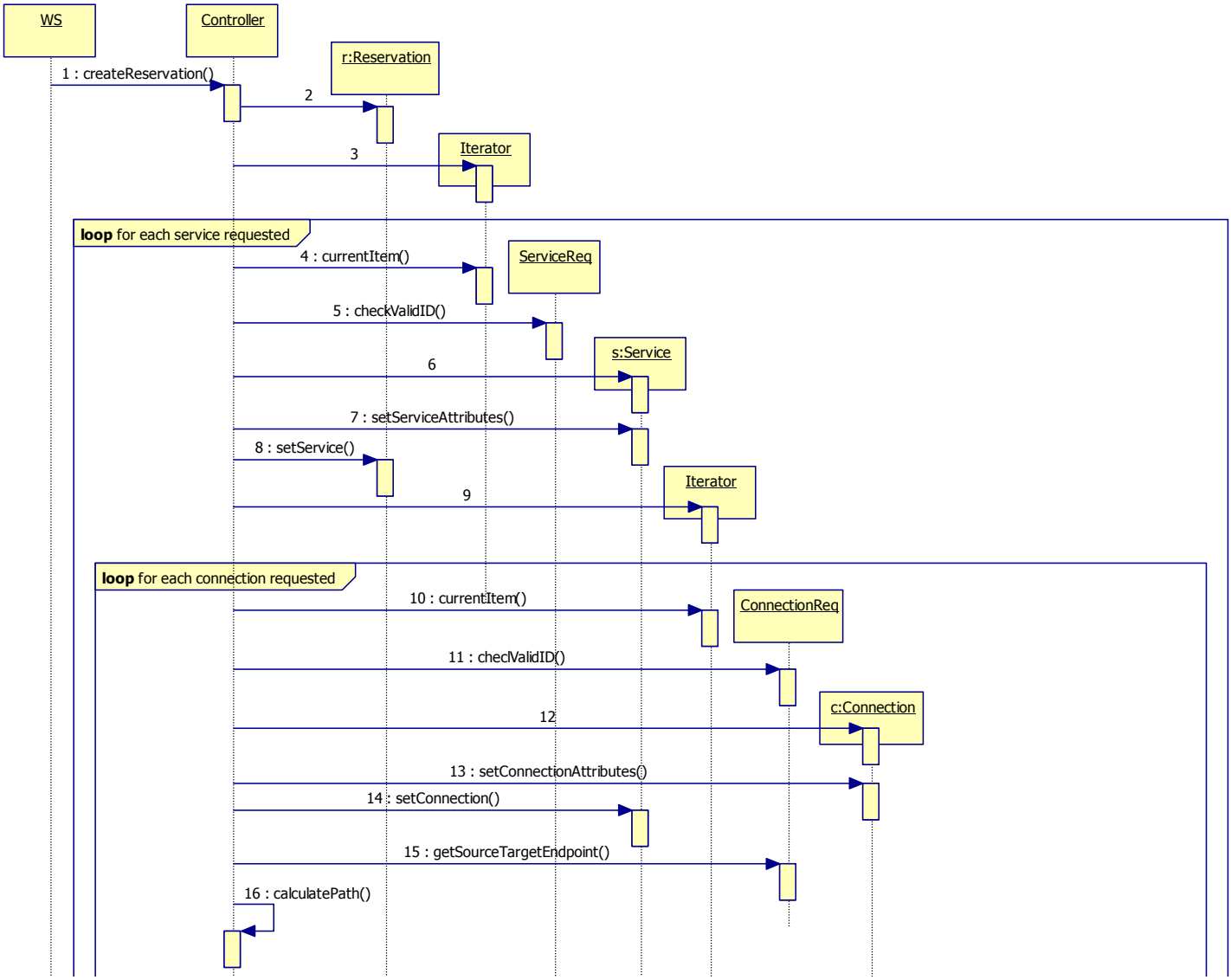
Cost: Θ(S * vi * C * vi * cp)

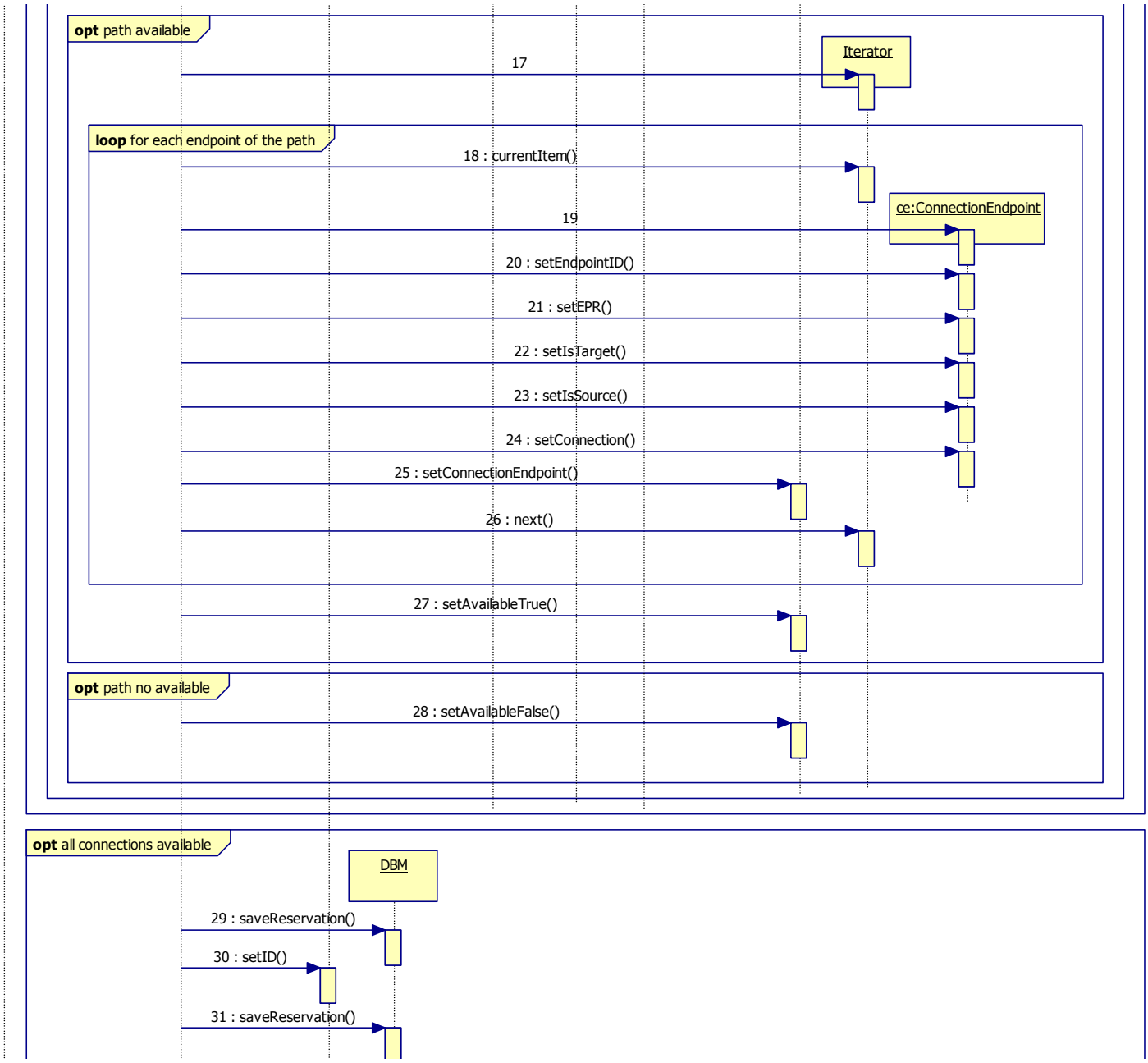
S = # services requested

vi = cost compute validEndpoint

C = # connections requested

cp = cost compute calculatePath




opt path available
Iterator
17
loop for each endpoint of the path
18 : currentItem()
ce:ConnectionEndpoint
19
20 : setEndpointID()
21 : setEPR()
22 : setIsTarget()
23 : setIsSource()
24 : setConnection()
25 : setConnectionEndpoint()
26 : next()
27 : setAvailableTrue()
opt path no available
28 : setAvailableFalse()
opt all connections available
DBM
29 : saveReservation()
30 : setID()
31 : saveReservation()

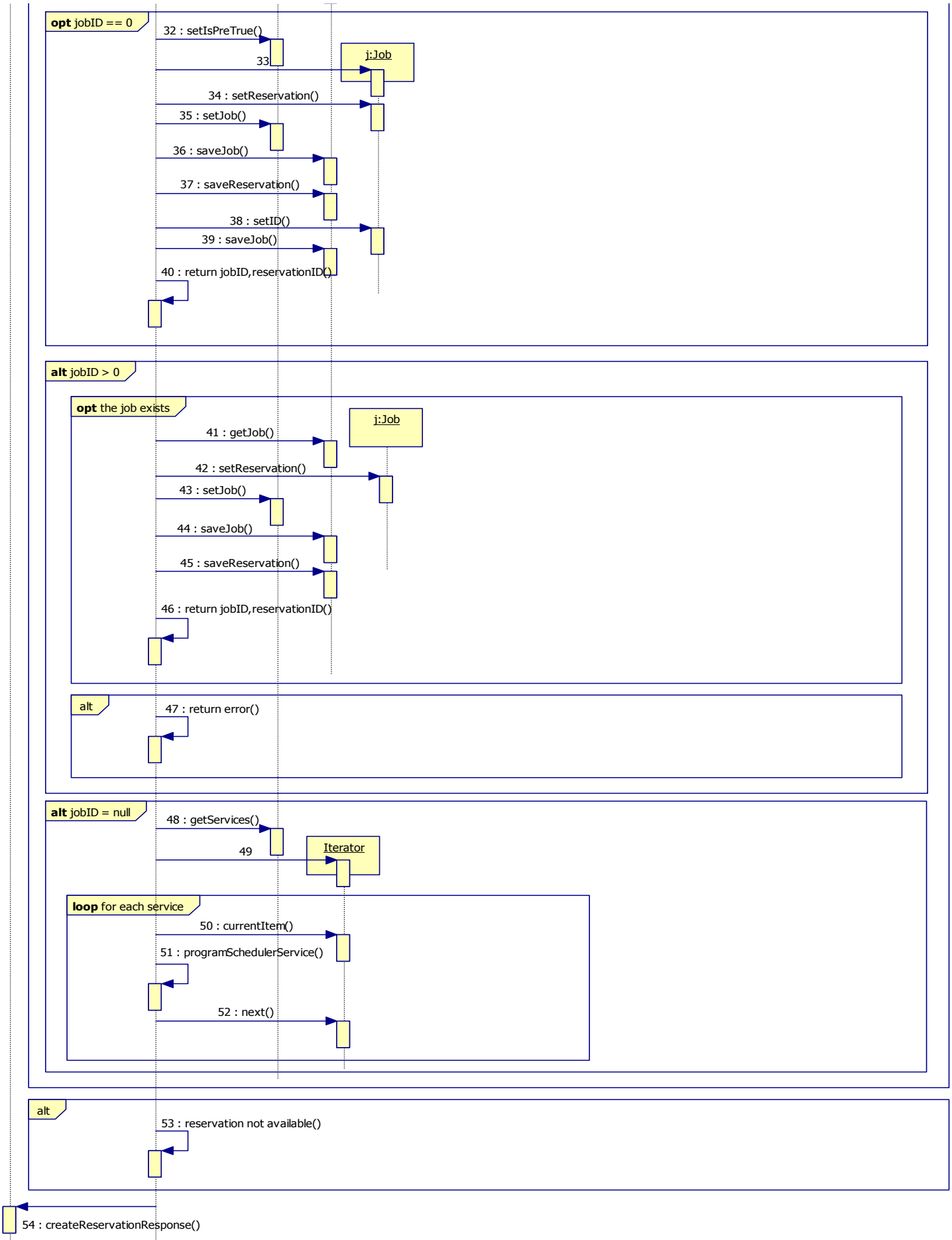
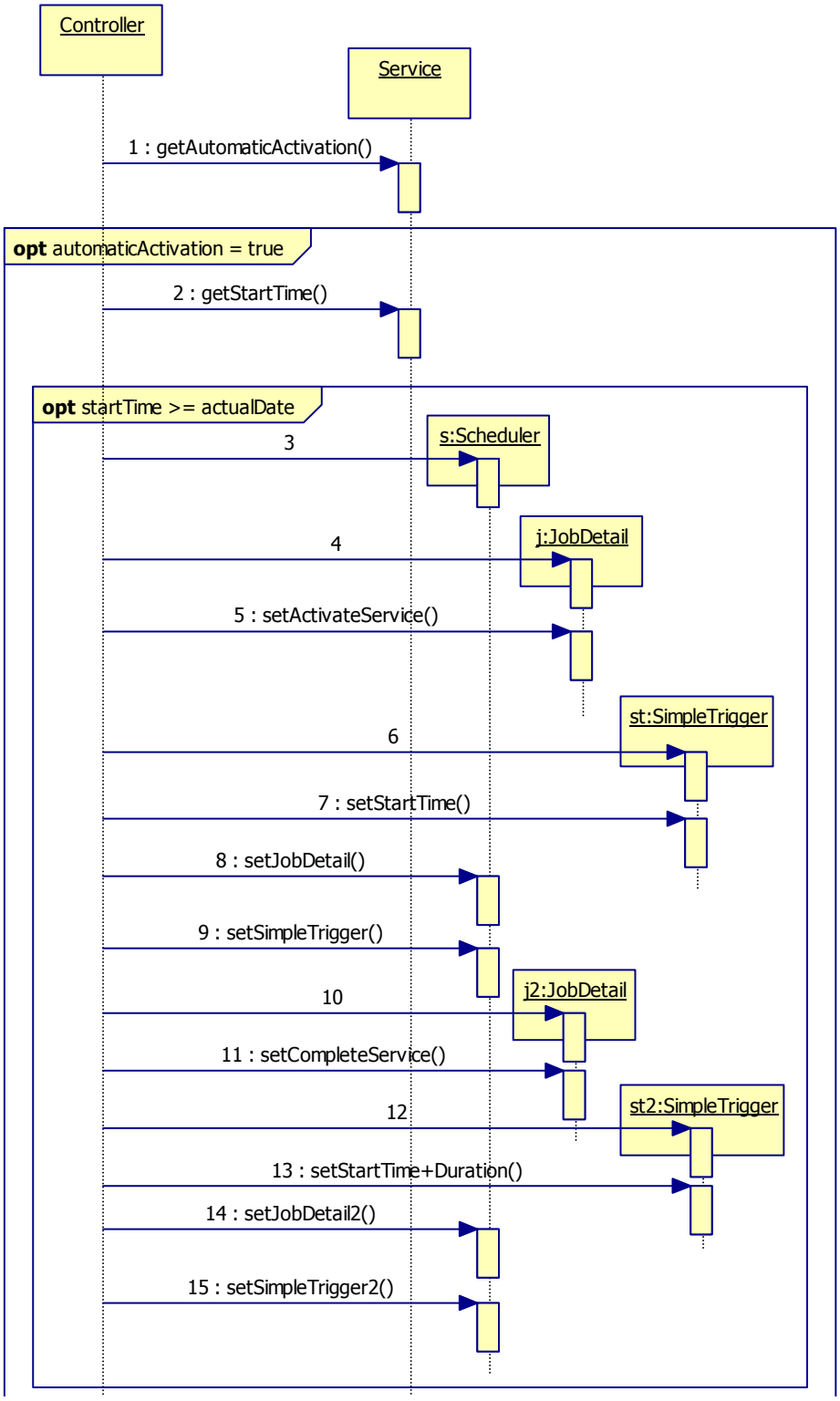


Figure 56. Create Reservation

programSchedulerService

A service is programmed to be activated and completed only if the automatic activation is set as true and if the start time requested is later than the actual date. To program the scheduler to activate a service, two JobDetail's and two SimpleTrigger's must be created. The first JobDetail activates the service, so the first SimpleTrigger is programmed at the start time of the service. The second jobDetail completes the service, so the second SimpleTrigger is programmed at the end time of the service.

Cost: Θ(1)



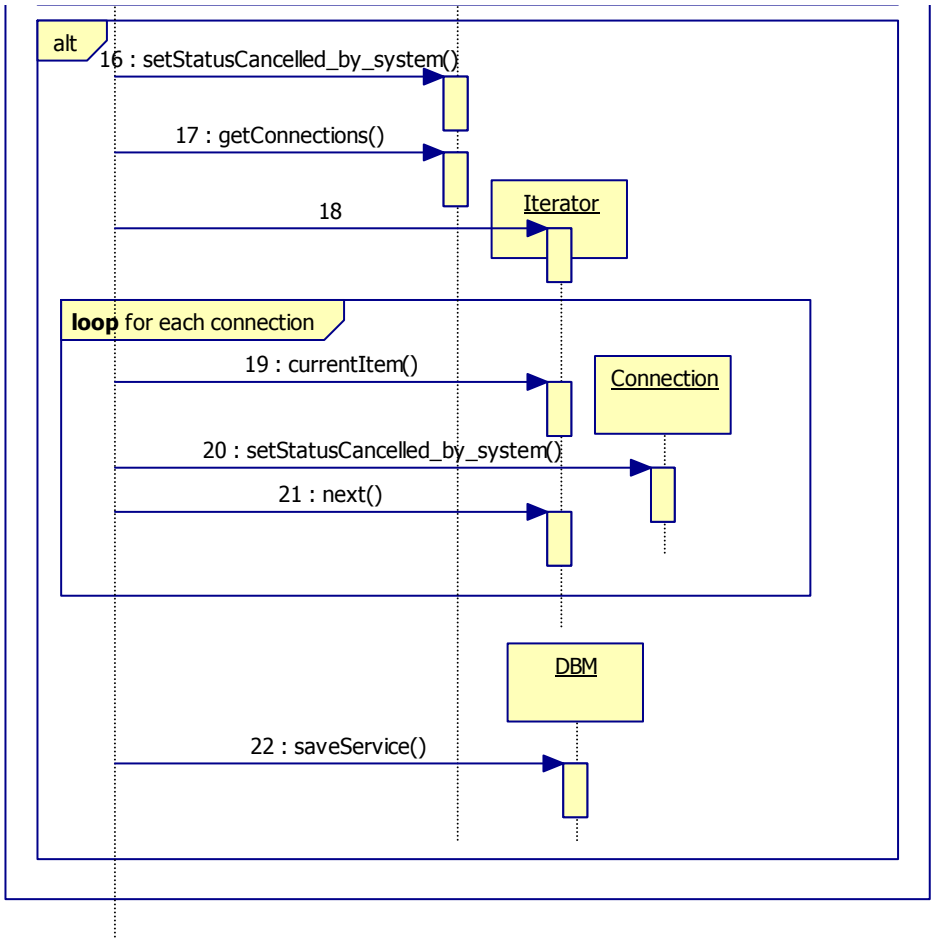


Figure 57. Program scheduler service

8.2.7. Cancel Reservation

The following picture shows the sequence diagram for the use case 'Cancel Reservation'. The WS receives a cancel reservation request and indicates to the controller which reservation must be cancelled. Firstly, the controller gets the reservation from the database. If the reservation exists, it checks the status of the services of the reservation. The connections of the services that have the status as 'active' are deleted physically. If the reservation is a pre-reservation, it is deleted from its job. Finally, the reservation is deleted from the database. If some error occurs, FALSE is returned.

Input: CancelReservation

- long reservationID
- AAAUserType
    - String userID
    - String password

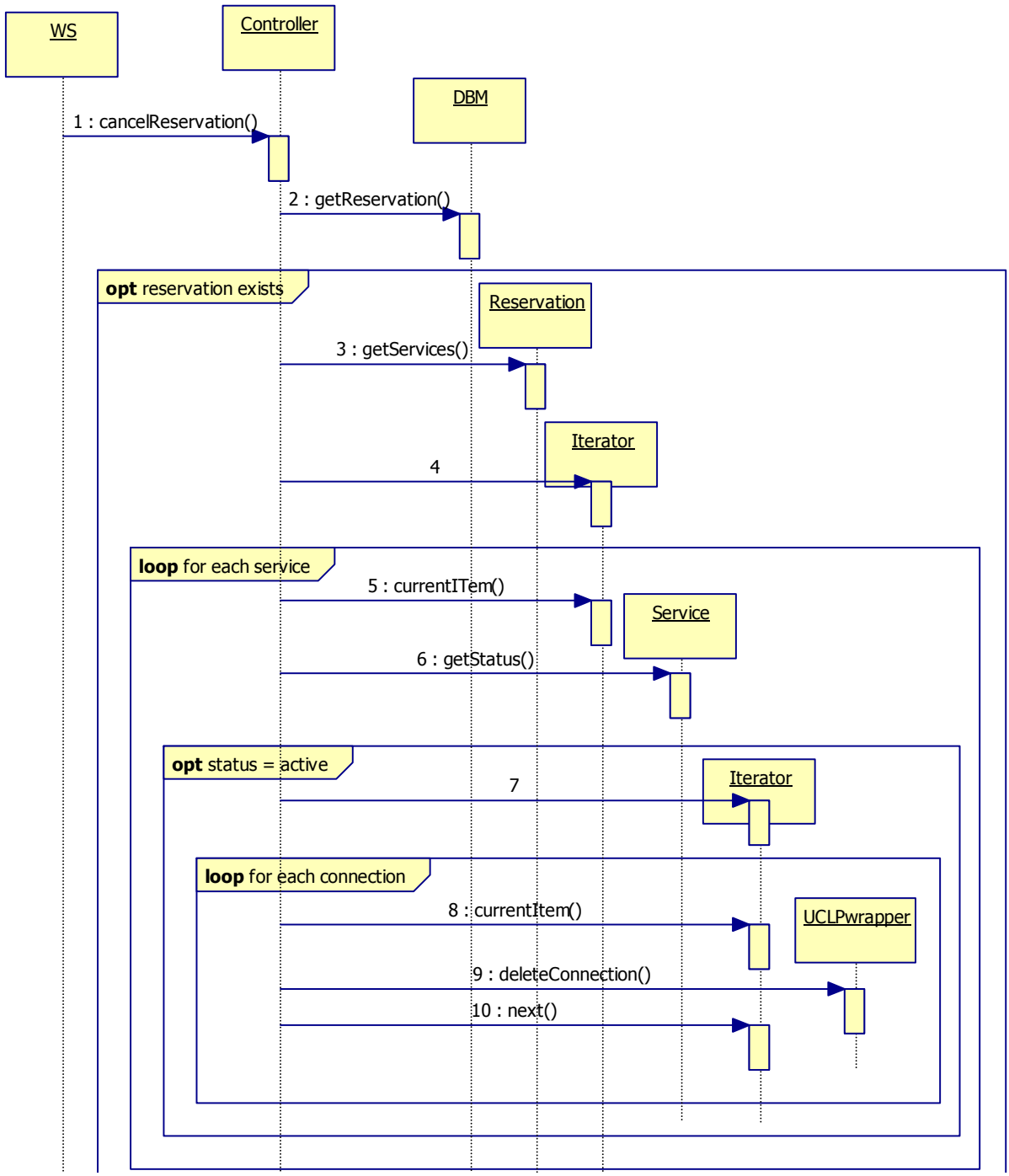
Output: CancelReservationResponse

- Boolean success

Cost: Θ(S * C)

S = # services

C = # connections


WS
Controller
DBM
1 : cancelReservation()
2 : getReservation()
opt reservation exists
Reservation
3 : getServices()
Iterator
4
loop for each service
5 : currentITem()
Service
6 : getStatus()
opt status = active
Iterator
7
loop for each connection
8 : currentItem()
UCLPwrapper
9 : deleteConnection()
10 : next()

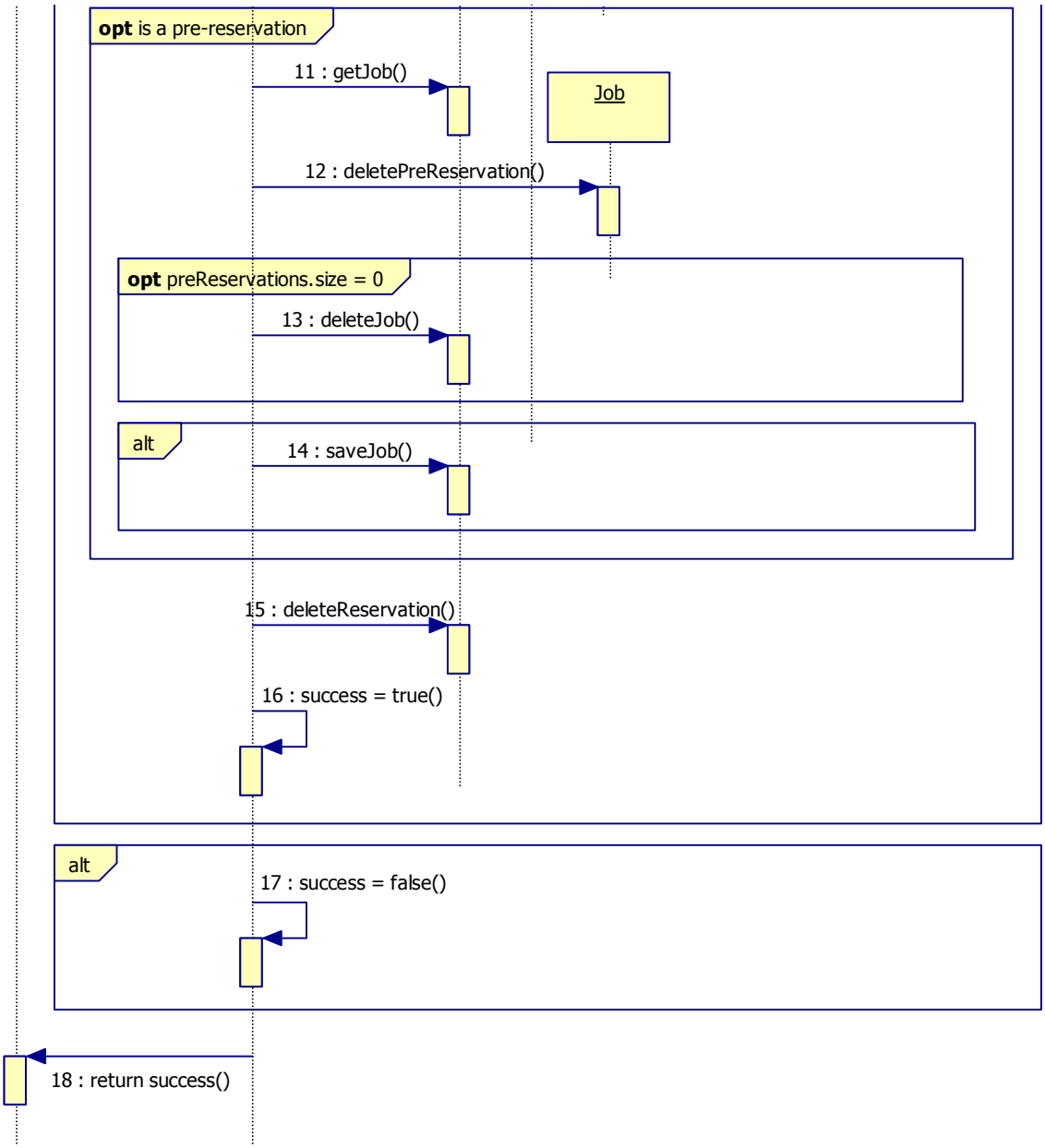


Figure 58. Cancel reservation

8.2.8. Get Status

The figure below shows the sequence diagram for the use case 'Get Status'. The WS indicates to the Controller which reservation and which services must be returned its status. If only the reservation identifier is given, the status of all the services of the reservation are returned; otherwise, if some service identifier is given, only the status of the services requested is returned. If the reservation or some services don't exist, NULL is returned.

Input: CancelReservation

- long reservationID
- int[] servicesID
- AAAUserType
    - String userID
    - String password

Output: CancelReservationResponse

- GetStatusResponseTypeServiceStatus [] serviceStatus
    - int serviceID
    - ReservationType typeOfReservation
    - FixedReservationConstraintType (optional)
        - Calendar startTime
        - Int duration
    - DeferrableReservationConstraintType (optional)
        - Calendar earliestStartTime
        - Int duration
        - Calendar deadline
    - MalleableReservationContraintType (optional)
        - Calendar earliestStartTime
        - Calendar deadline
    - Boolean automaticActivation
    - StatusType status
    - ConnectionStatusType[] connections
        - int connectionID
        - EndpointType source
        - EndpointType[] target
        - Int directionality
        - StatusType status

Cost: Θ(S)

S = # services requested or # services of the reservation

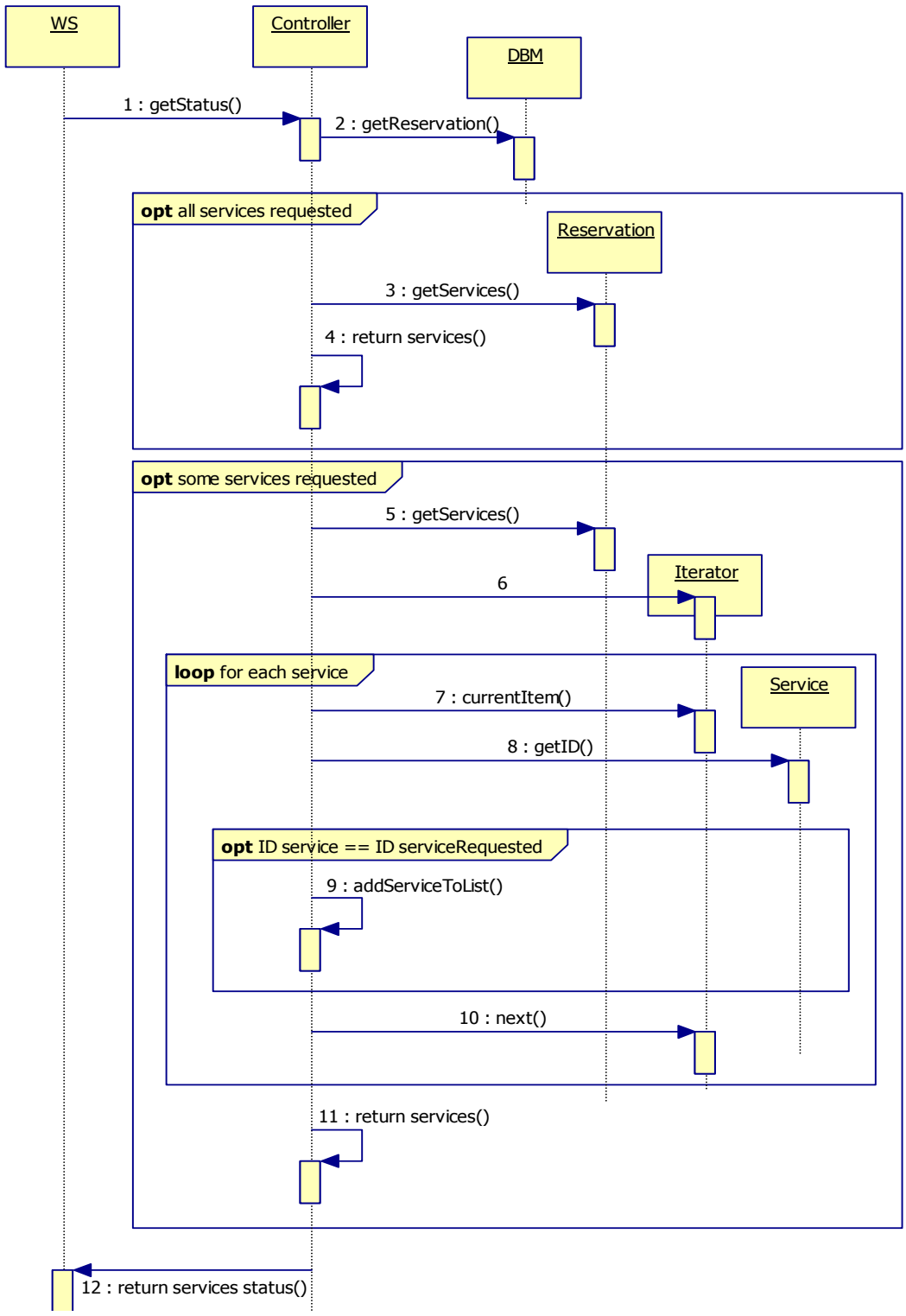


Figure 59. Get Status

8.2.9. Complete Job

The next figure shows the sequence diagram of the use case 'Complete Job'. The WS receives the identifier of the job to be completed and indicates it to the Controller. The Controller checks if this job exists. If the job doesn't exist FALSE is returned. Otherwise, for each pre-reservation of the job, the isPre attribute is set as false; the pre-reservation is disassociated from the job and for each service of the pre-

reservation, if the start time is earlier than the present time, the status attribute is set as cancelled_by_system. The services that have the start time later than the present time are programmed to be activated. Finally, the reservation is saved and the job deleted, and if everything has gone right, TRUE is returned.

Input: CompleteJob

- long jobID
- AAAUserType
    - String userID
    - String password

Output: CompleteJobResponse

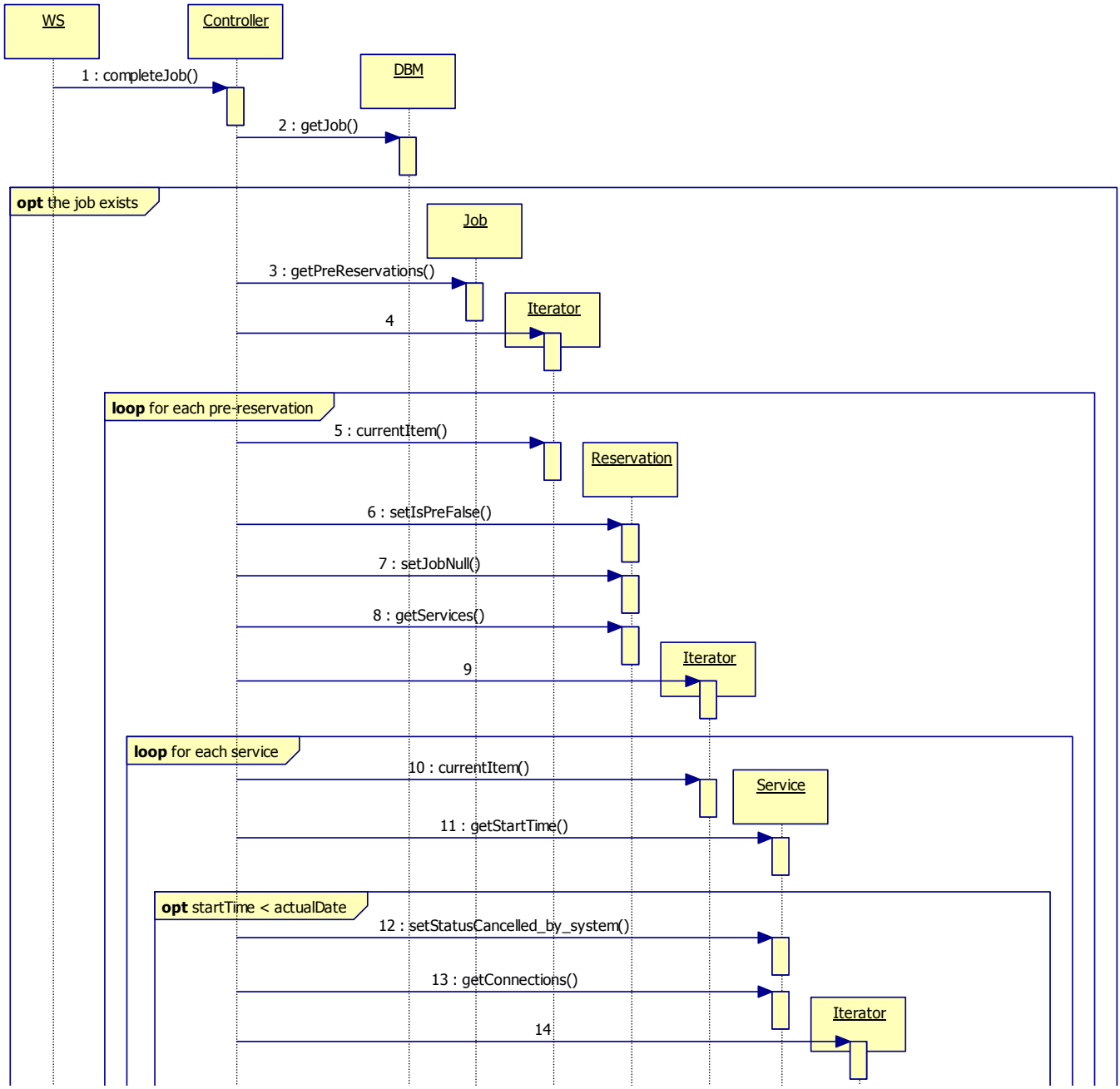
- Boolean success

Cost: Θ(PR * S * C)

PR = # pre-reservations of the job

S = # services of the pre-reservation

C = # connections of the service


WS
Controller
DBM
1 : completeJob()
2 : getJob()
opt the job exists
Job
3 : getPreReservations()
Iterator
4
loop for each pre-reservation
5 : currentItem()
Reservation
6 : setIsPreFalse()
7 : setJobNull()
8 : getServices()
Iterator
9
loop for each service
10 : currentItem()
Service
11 : getStartTime()
opt startTime < actualDate
12 : setStatusCancelled_by_system()
13 : getConnections()
Iterator
14

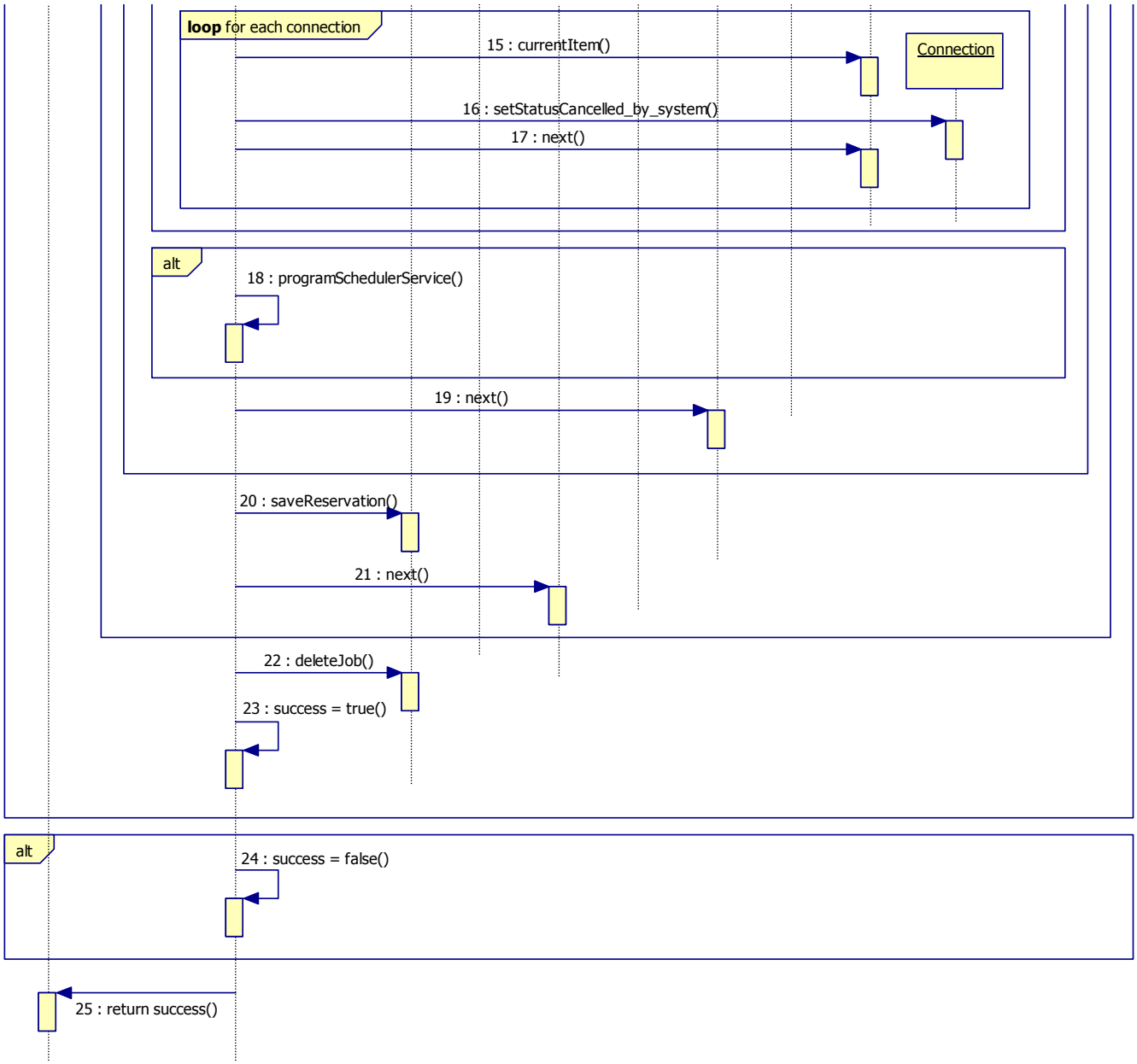


Figure 60. Complete job

8.2.10. Cancel Job

The figure number 59 shows the sequence diagram of the use case ‘Cancel Job’. The WS receives the identifier of the job to be cancelled and indicates it to the Controller. The Controller checks if this job exists. If the job doesn’t exist FALSE is returned. Otherwise, each pre-reservation of the job is deleted from the database and also the job. If everything goes OK, TRUE is returned.

Input: CancelJob

- long jobID
- AAAUserType
    - String userID
    - String password

Output: CancelJobResponse

- Boolean success

Cost: Θ(PR)

PR = # pre-reservations of the job

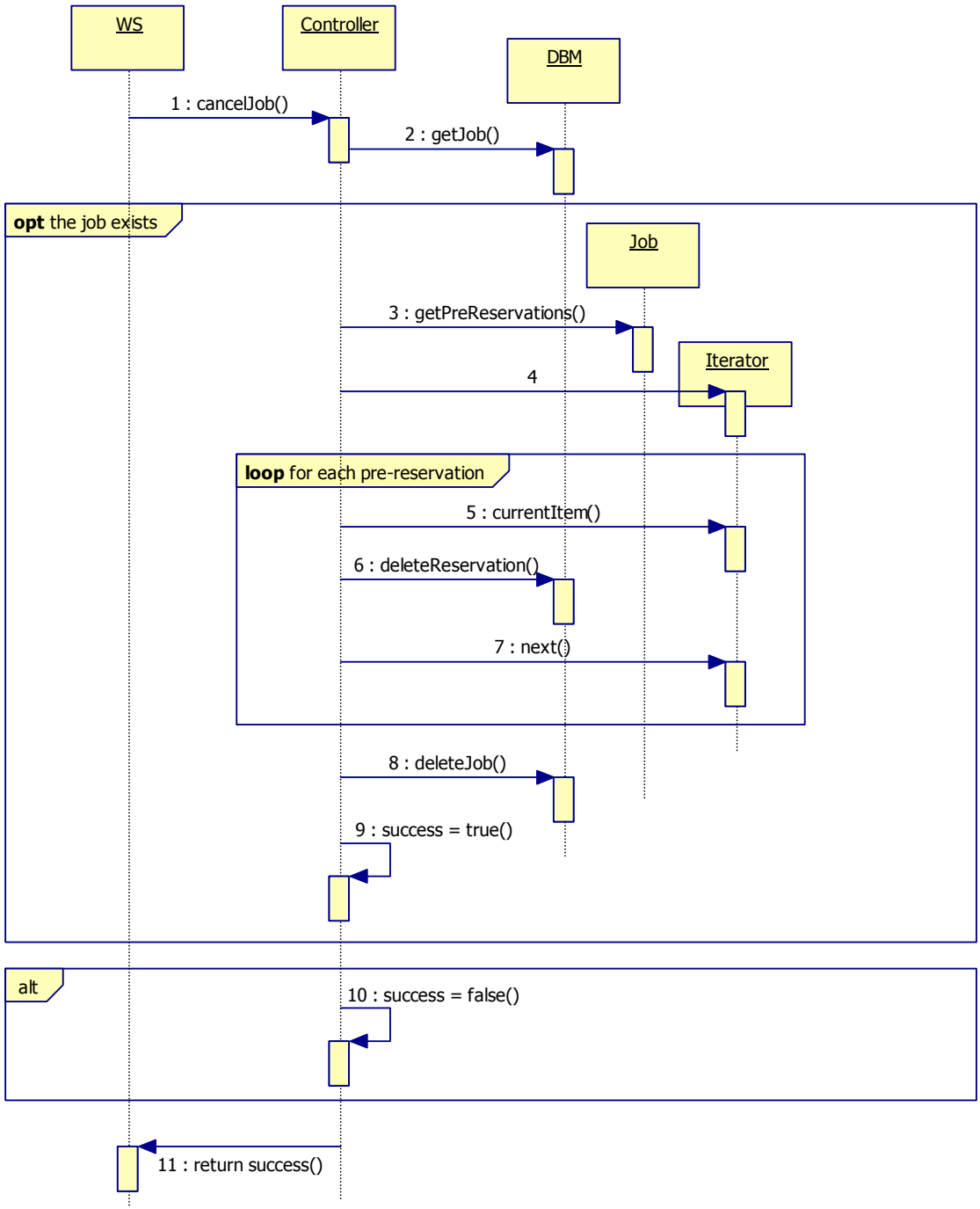


Figure 61. Cancel job

8.2.11. Activate

In the figure below the sequence diagram for the use case 'Activate' is shown. The WS indicates to the Controller which service must be activated. The controller gets the service from the database, if this service comes from a pre-reservation, FALSE is returned. If the status of the service is pending and the start time is latter than the present time, the scheduler is programmed to activate the service in its start time. If the start time has passed, an alternative start time as soon as possible is calculated; if exists an alternative start time to activate the reservation, this time is set as the new start time, the automatic activation is set as true and the scheduler is programmed to activate the service in this new start time. If some error occurs, FALSE is returned, otherwise TRUE is returned.

Input: Activate

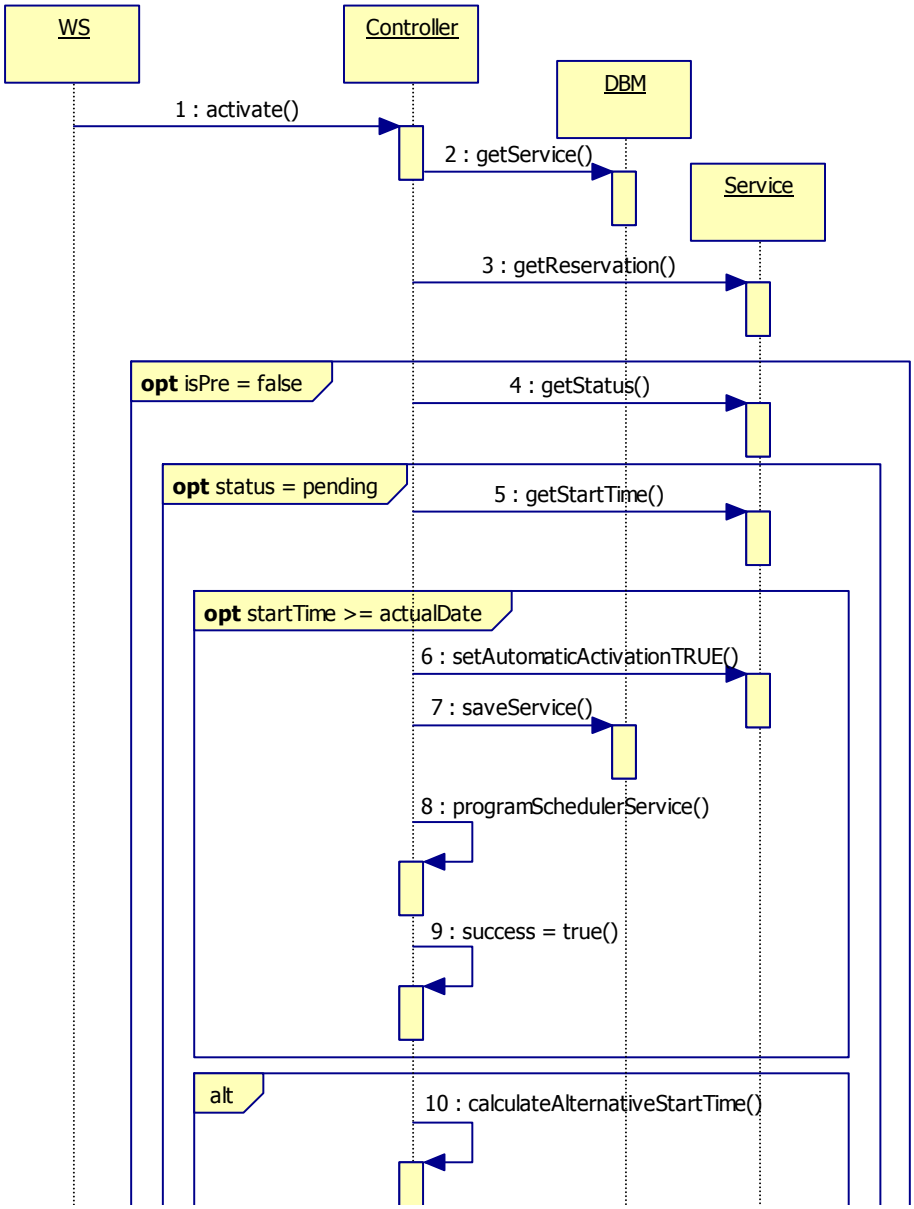
- Long reservationID
- Int serviceID
- AAAUserType
    - String userID
    - String password

Output: CancelJobResponse

- Boolean success

Cost: O(ca)

ca = cost compute calculateAlternativeStartTime


WS
Controller
DBM
Service
1 : activate()
2 : getService()
3 : getReservation()
opt isPre = false
4 : getStatus()
opt status = pending
5 : getStartTime()
opt startTime >= actualDate
6 : setAutomaticActivationTRUE()
7 : saveService()
8 : programSchedulerService()
9 : success = true()
alt
10 : calculateAlternativeStartTime()

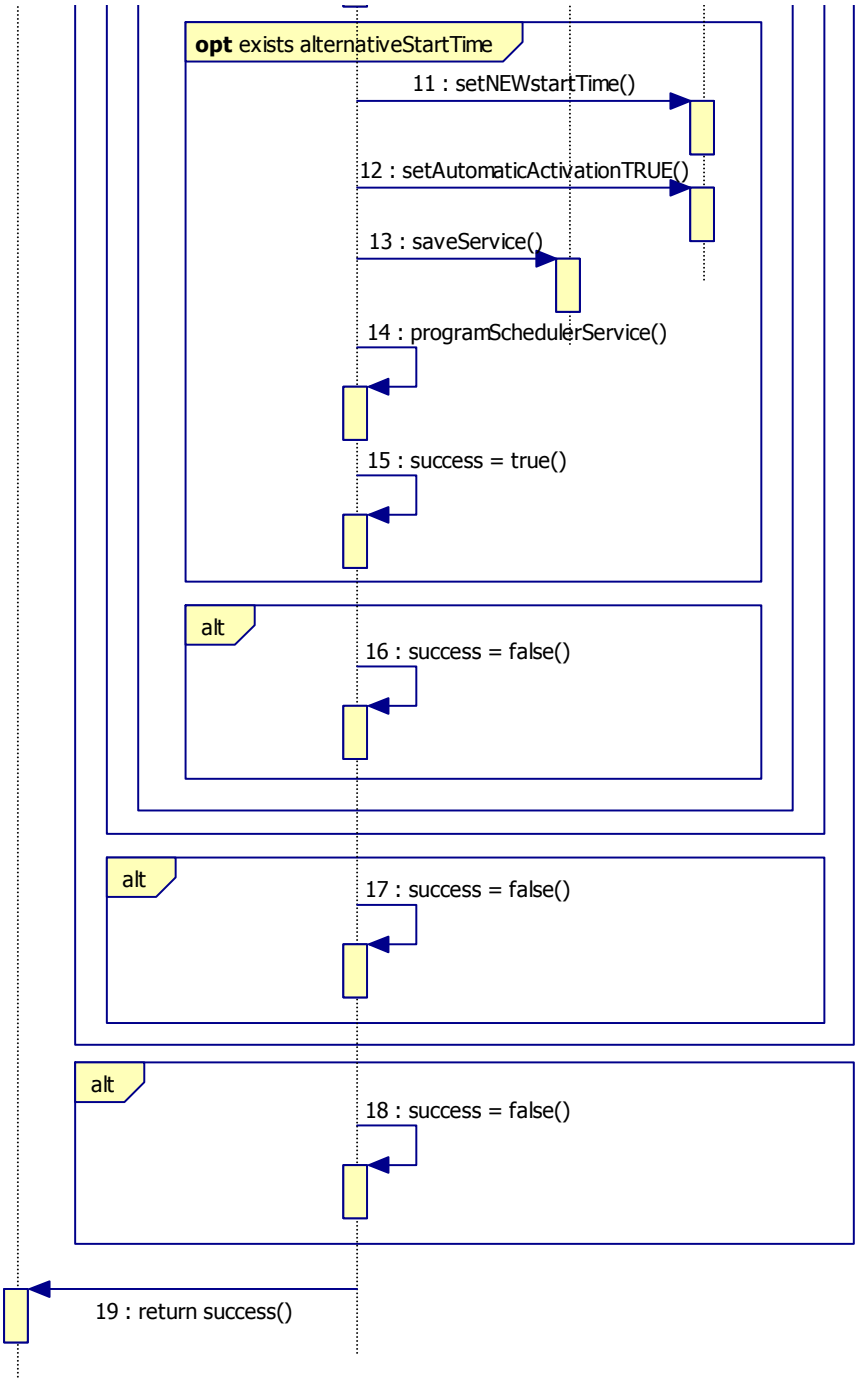


Figure 62. Activate

8.2.12. Activate Automatically

In the following figure, the sequence diagram for the use case 'Activate Automatically' is shown. When the start time of a service arrives, the scheduler indicates it to the controller, who sets the status of the service and its connections as active, and calls the UCLP system to activate physically the connections. Finally the service is stored in the database.

Input: ActivateAutomatically

- long reservationID
- int serviceID

Output: ActivateAutomaticallyResponse

- Boolean success

Cost: Θ(C)

C = # connections of the service

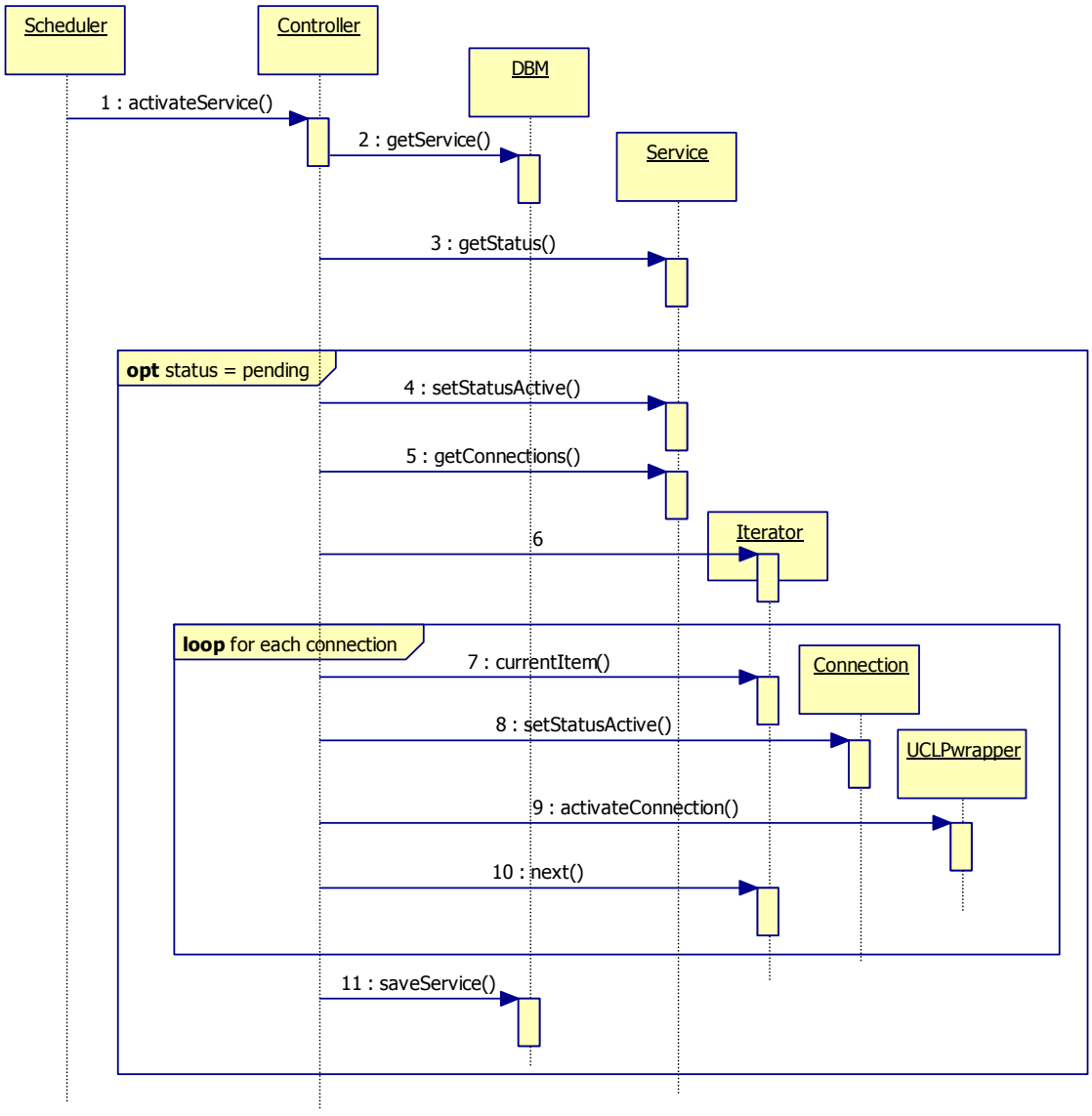


Figure 63. Activate automatically

8.2.13. Complete Service

In the following figure, the sequence diagram for the use case 'Complete Service' is shown. When the end time of a service arrives, the scheduler indicates it to the controller, who sets the status of the service and its connections as completed, and calls the UCLP system to delete physically the connections. Finally the service is stored in the database.

Input: CompleteService

- long reservationID
- int serviceID

Output: CompleteServiceResponse

- Boolean success

Cost: Θ(C)

- C = # connections of the service

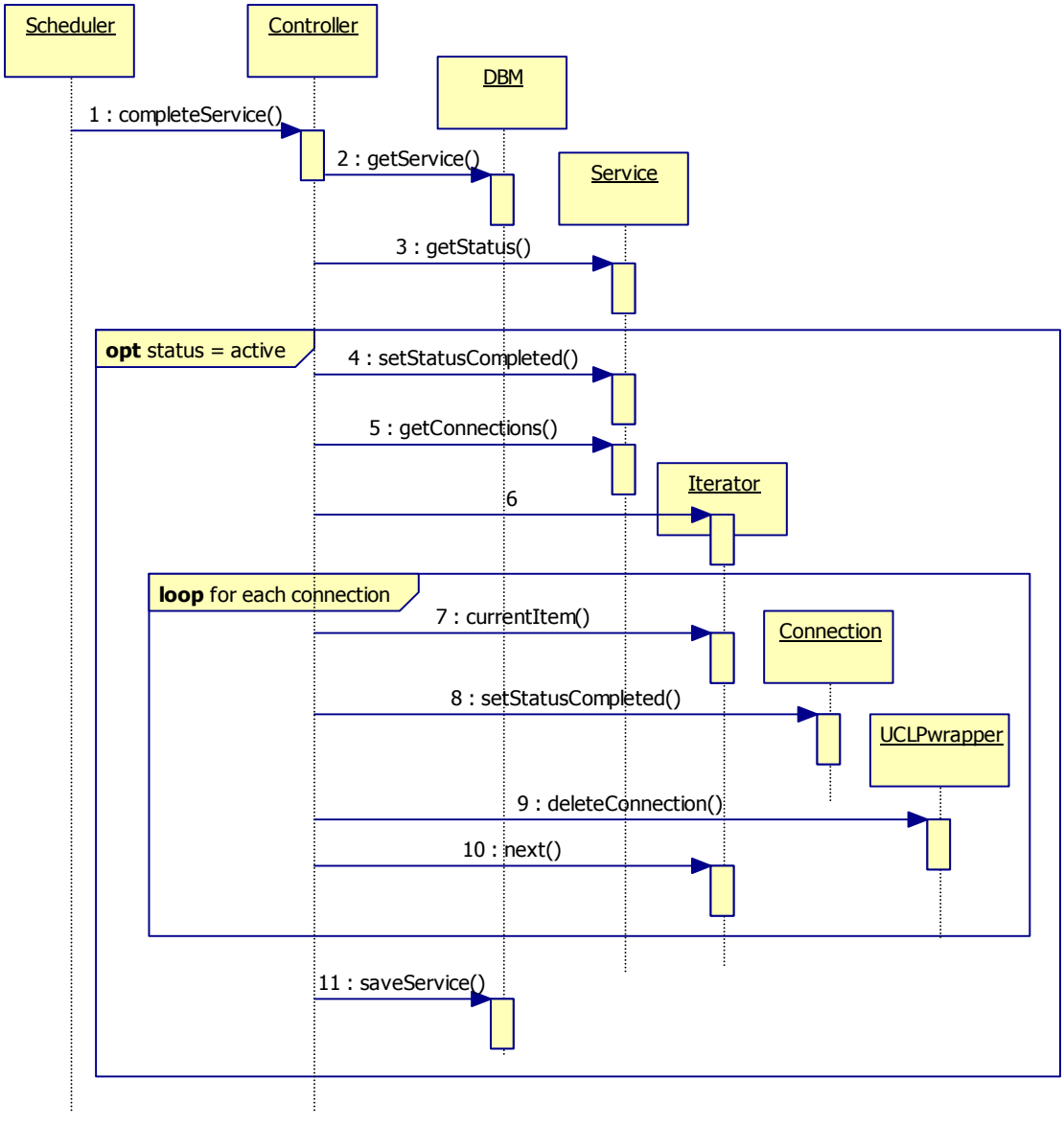


Figure 64. Complete Service

8.2.14. Reconfigure System

In the figure below, the sequence diagram for the use case 'Reconfigure System' is shown. If the system has fallen down, when it turns up again, the following verifications must be done:

For each reservation stored in the database:

- If it is a permanent reservation, for each connection of each service of this reservation, it must be checked if a new route for the connection has to be calculated because some node, link or endpoint has been removed or added in the network. If the path of the connection must be changed, a new path for the connection is calculated. Then, for each service:
    - If its status was 'active' and its end time has not arrived; the scheduler must be reprogrammed to complete the service in its end time. Otherwise, if the end time has passed while the system was down, the Controller contacts the UCLP system to delete physically the connections and the status of the service and the connections is set as 'cancelled_by_system'.
    - If the status was 'pending', the end time has not arrived and the reservation was marked as activate automatically, the Controller contacts the UCLP system to create the connections physically, the status of the service and the connections is set as 'active' and the scheduler is programmed to complete the service in its end time. If the end time has arrived, the status of the service and the connections is set as 'cancelled_by_system'. And finally, if the start time has not arrived, the scheduler is programmed to activate the service in its end time and to complete it in its end time.
- If the reservation is a pre-reservation, for each service, if the start time is earlier than the present time, the status of the service and its connections is set as 'cancelled_by_system'. Otherwise, for each connection is checked if a new route has to be calculated because the network has changed. If the path of the connection has been modified, a new path is calculated.

Once these verifications have been done, the reservation is updated in the database.

Finally, the connectionEndpoint's that have to be deleted because its endpoints have been removed from the network are deleted from the database.

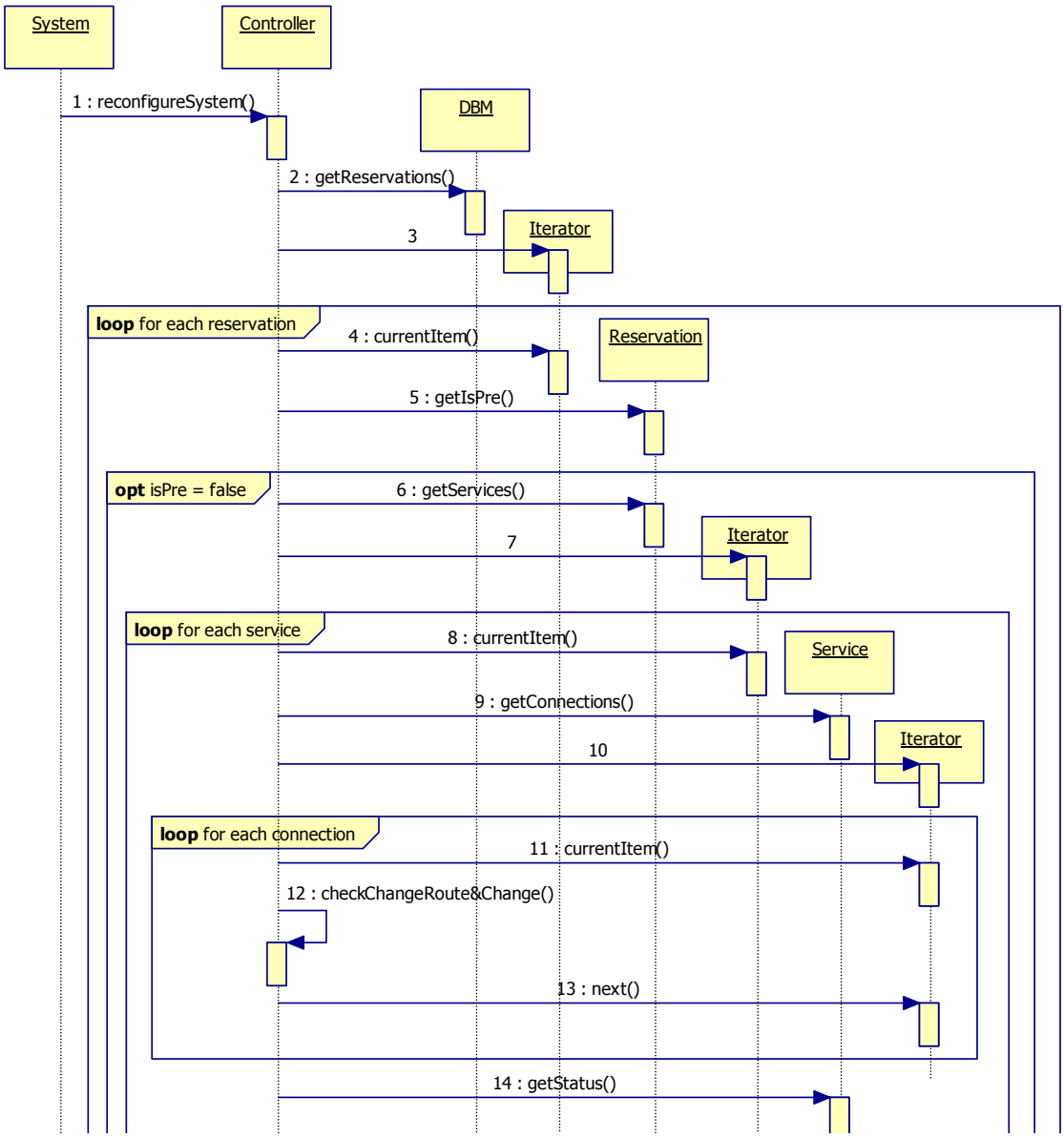
Cost: O(R * S * C * cc)

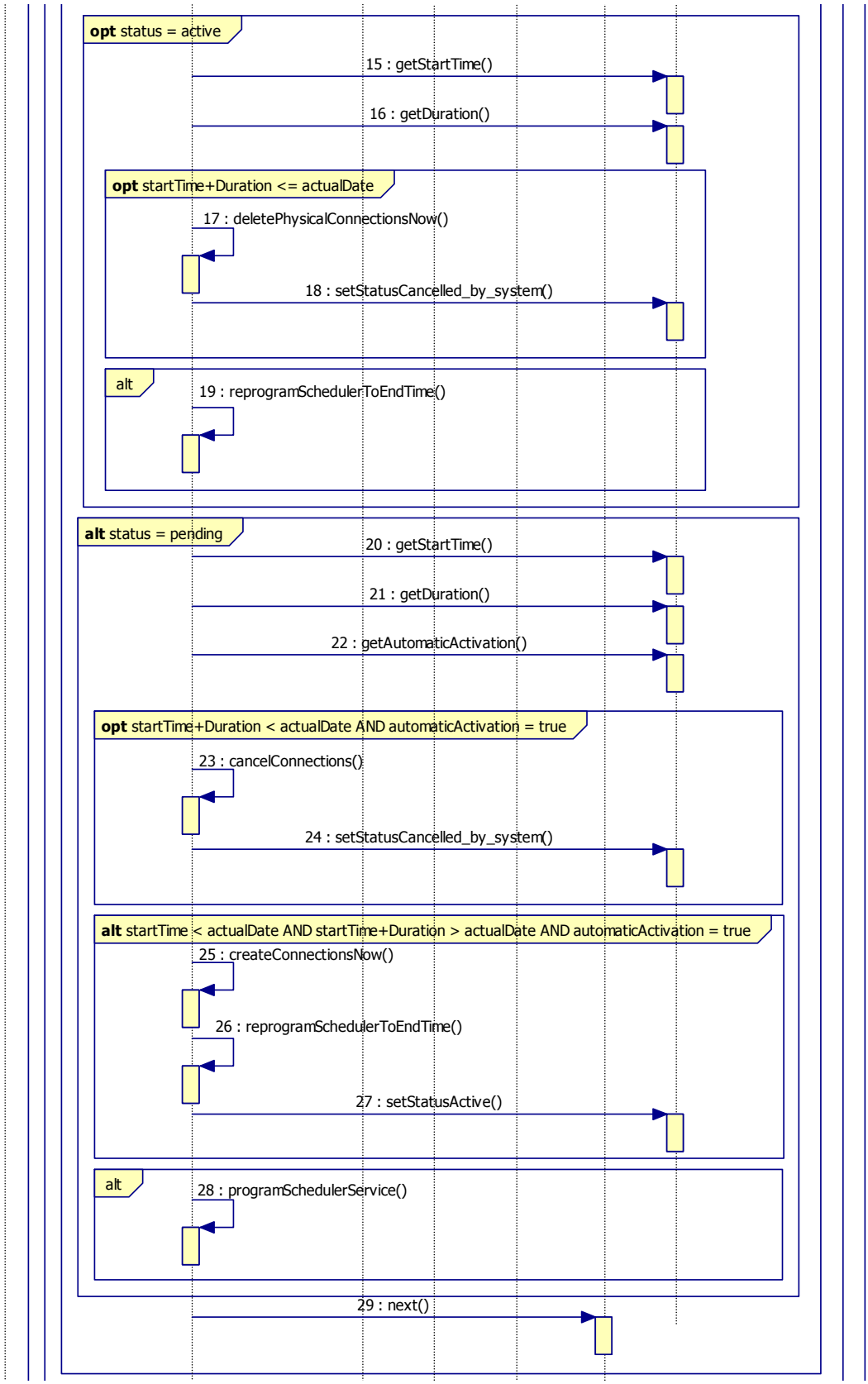
R = # reservations stored in the DB

S = # services of the reservation

C = # connections of the service

Cc = cost compute checkChangeRoute&Change


System
Controller
DBM
1 : reconfigureSystem()
2 : getReservations()
Iterator
3
loop for each reservation
4 : currentItem()
Reservation
5 : getIsPre()
opt isPre = false
6 : getServices()
Iterator
7
loop for each service
8 : currentItem()
Service
9 : getConnections()
Iterator
10
loop for each connection
11 : currentItem()
12 : checkChangeRoute&Change()
13 : next()
14 : getStatus()


opt status = active
15 : getStartTime()
16 : getDuration()
opt startTime+Duration <= actualDate
17 : deletePhysicalConnectionsNow()
18 : setStatusCancelled_by_system()
alt
19 : reprogramSchedulerToEndTime()
alt status = pending
20 : getStartTime()
21 : getDuration()
22 : getAutomaticActivation()
opt startTime+Duration < actualDate AND automaticActivation = true
23 : cancelConnections()
24 : setStatusCancelled_by_system()
alt startTime < actualDate AND startTime+Duration > actualDate AND automaticActivation = true
25 : createConnectionsNow()
26 : reprogramSchedulerToEndTime()
27 : setStatusActive()
alt
28 : programSchedulerService()
29 : next()

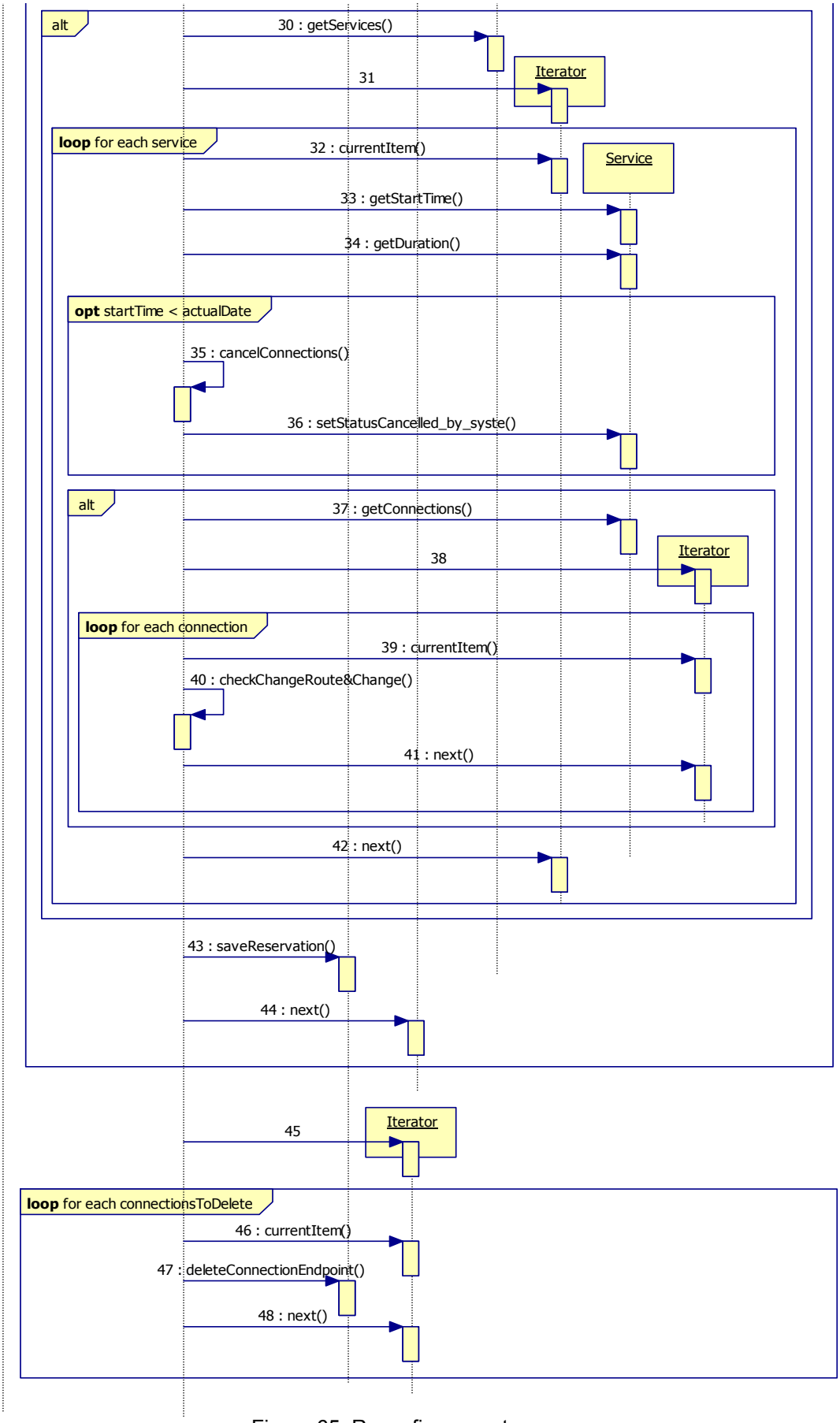


Figure 65. Reconfigure system

checkChangeRoute&Change

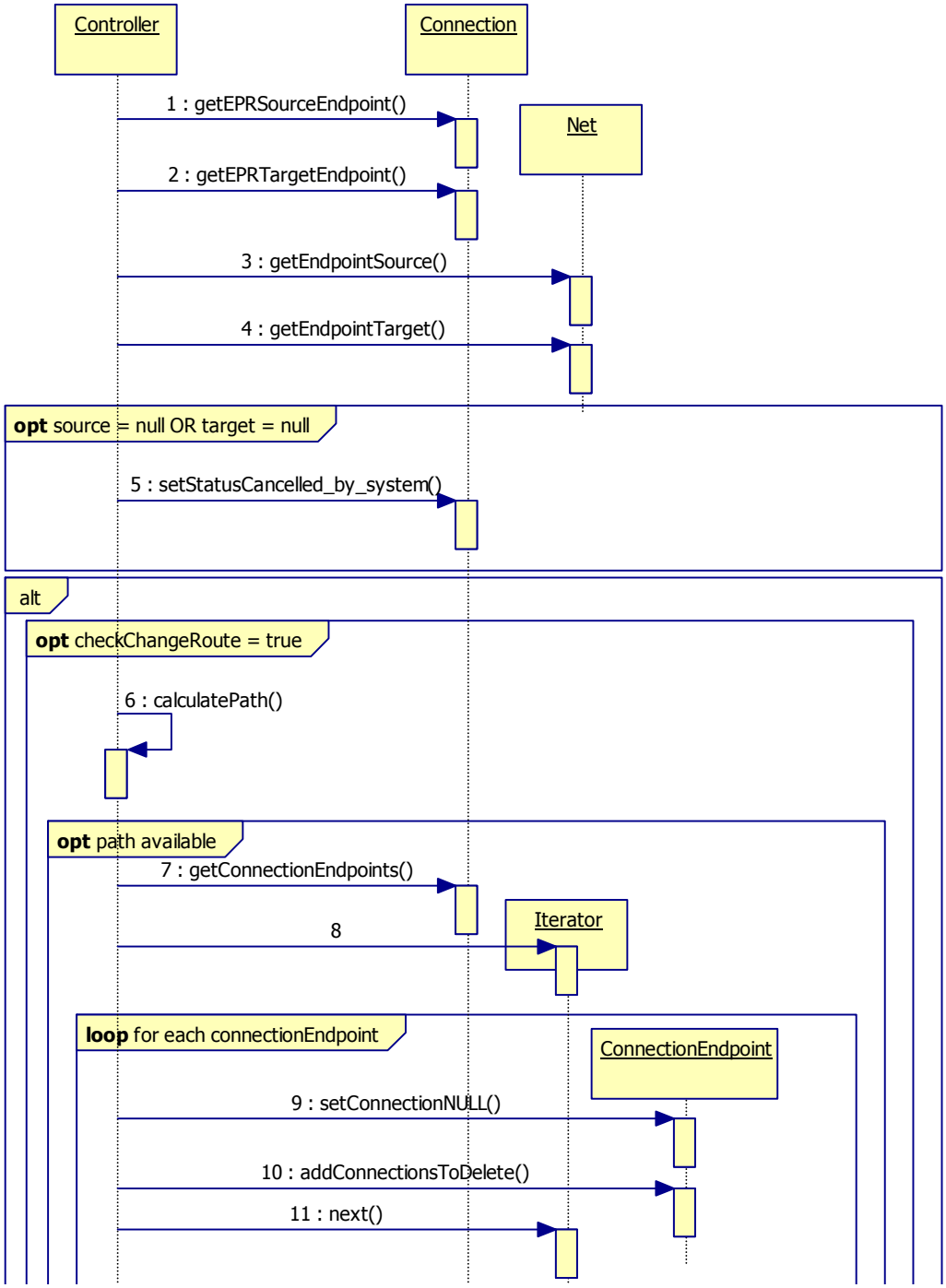
First, the controller checks if the source and target endpoint of the connection already exist in the network. If some of them have been removed, the status of the connection is set as ‘cancelled_by_system’. If they already exists, it is checked if the route of the connection must be changed. If the route must be changed because some link, endpoint or node has been removed, a new path for the connection is calculated. If a possible path is available, all the old ConnectionEndpoint’s of the connection are set in the list ‘connectionsToDelete’, and for each endpoint of the new route, a new connectionEndpoint is created and associated to the connection. If no possible path is available, the status of the connection is set as ‘cancelled_by_system’.

Cost: O(cr * cp * C)

Cr = cost compute checkChangeRoute

Cp = cost compute calculatePath

C = # connections


Controller
Connection
Net
1 : getEPRSourceEndpoint()
2 : getEPRTargetEndpoint()
3 : getEndpointSource()
4 : getEndpointTarget()
opt source = null OR target = null
5 : setStatusCancelled_by_system()
alt
opt checkChangeRoute = true
6 : calculatePath()
opt path available
7 : getConnectionEndpoints()
Iterator
8
loop for each connectionEndpoint
ConnectionEndpoint
9 : setConnectionNULL()
10 : addConnectionsToDelete()
11 : next()

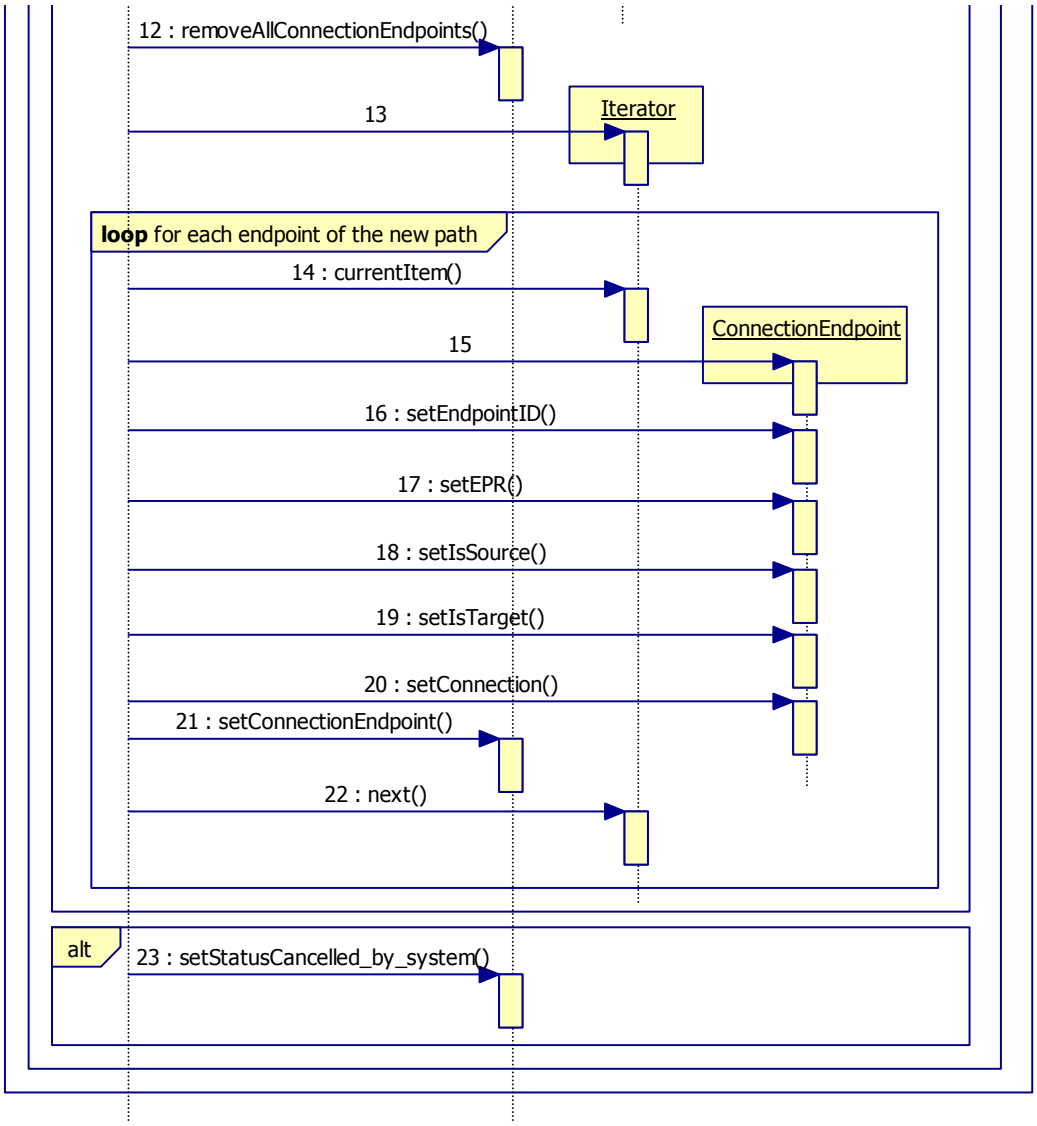


Figure 66. Check change route and change

checkChangeRoute

To verify if the network has changed, for each endpoint of the connection it is checked if an endpoint in the network with the same identifier and with the same EPR exists. If no endpoint with the same identifier exists, or with the same identifier but not with the same EPR, it means that the network has change and so, a new path for the connection must be calculated.

Cost: Θ(CE)

CE = # connectionEndpoint's

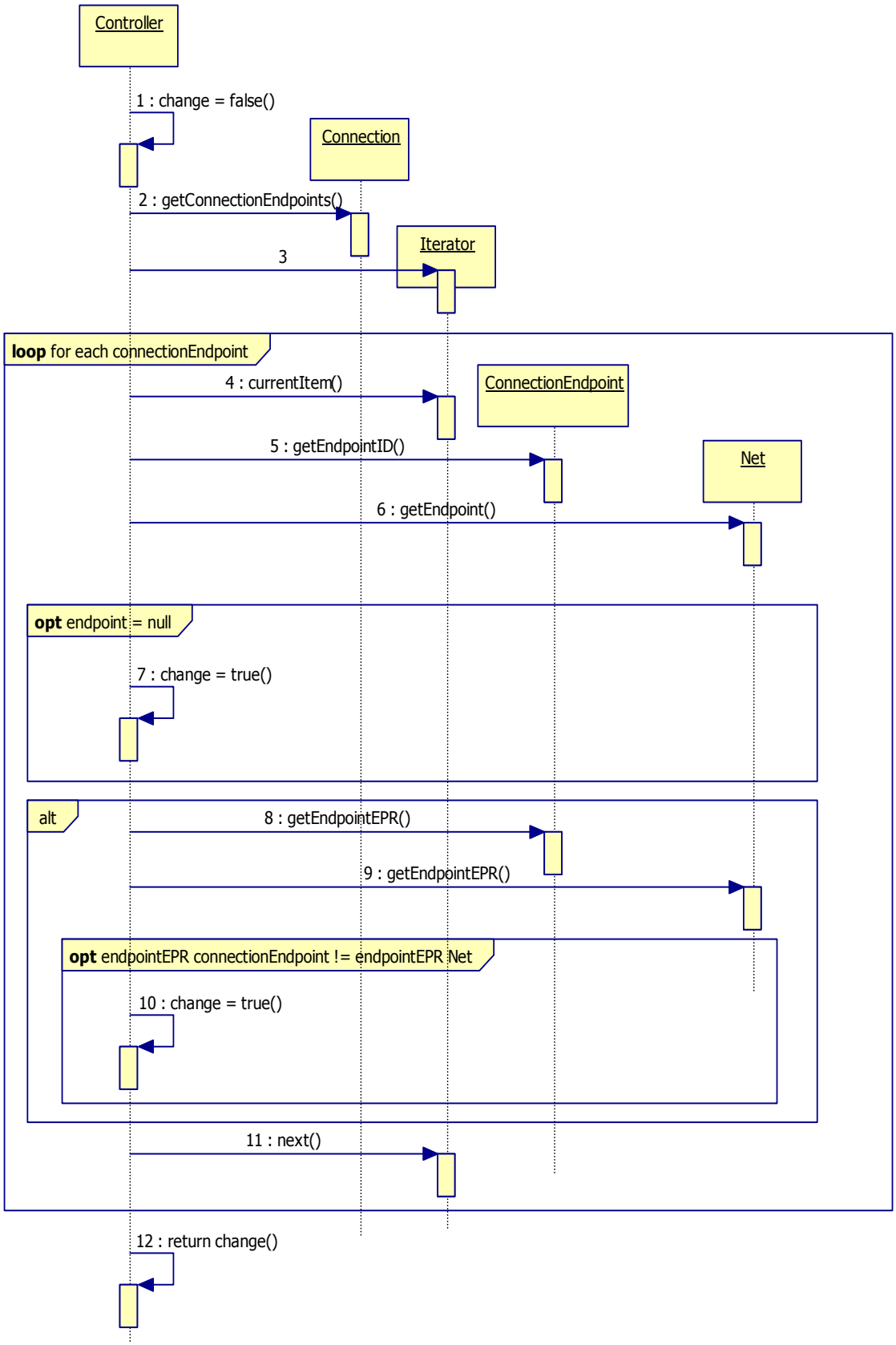


Figure 67. Check change route

deletePhysicalConnectionsNow

For each connection of the service, the controller contacts the UCLP system to delete the connection and sets the status of the connection as 'cancelled_by_system'.

Cost: Θ(C)

C = # connections

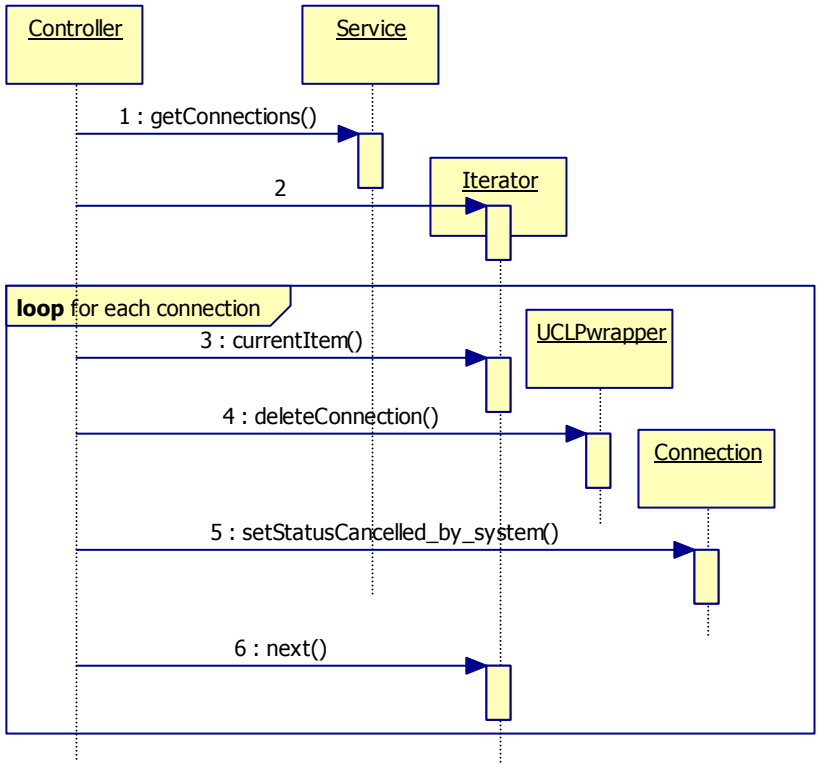


Figure 68. Delete physical connections now

reprogramSchedulerToEndTime

The scheduler is programmed by the controller to complete the service in its end time.

Cost: Θ(1)

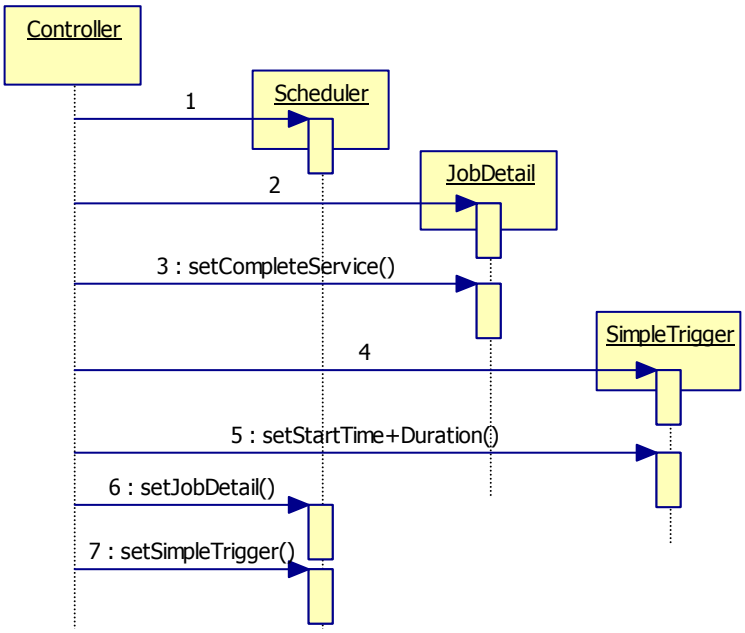


Figure 69. Reprogram scheduler to end time

cancelConnections

For each connection of the service, the controller sets the status of the connections as 'cancelled_by_system'.

Cost: Θ(C)

C = # connections

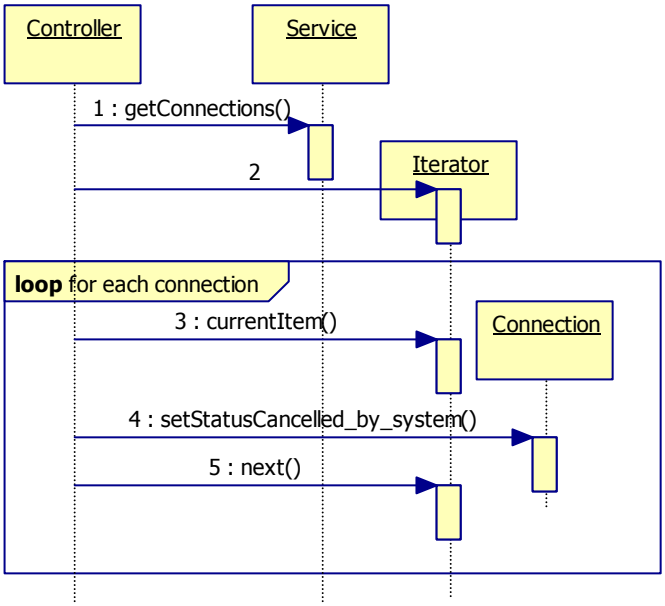


Figure 70. Cancel connections

createConnectionsNow

For each connection of the service, the controller contacts the UCLP system to create them physically and sets its status as 'active'.

Cost: Θ(C)

C = # connections

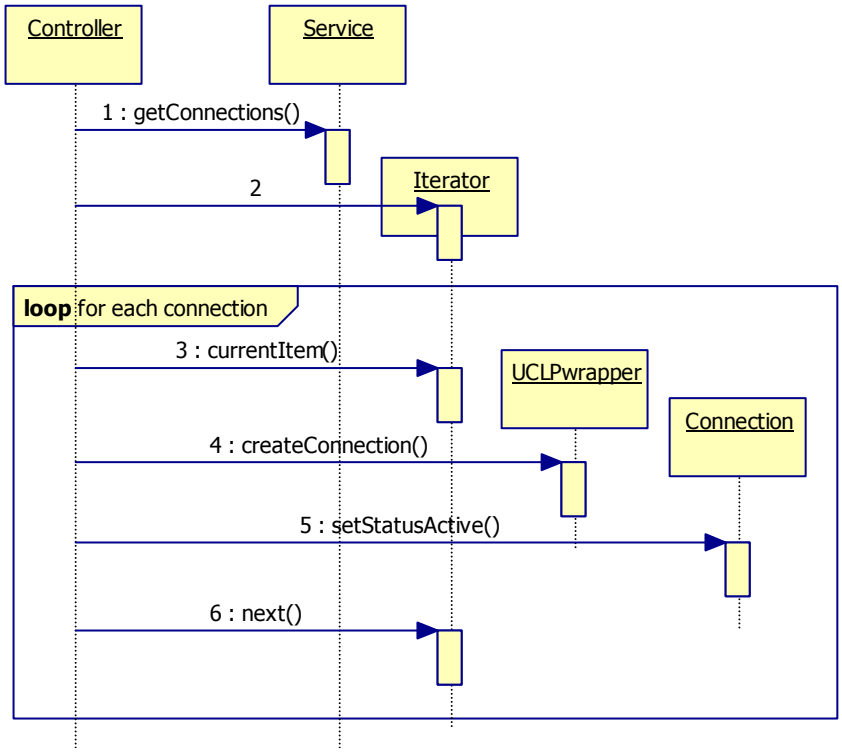


Figure 71. Create connections now

8.3. Design of the Data Management Layer

One of the main requirements of this system is the data persistence. To obtain this persistence a database is required in order to store data and get the data stored previously. This section describes the design done for the database of the system from the conceptual model (section 7.2).

8.3.1. Conceptual Design

The conceptual design is obtained by transforming the classes and relations of the conceptual model in an Entity-Relationship diagram. As in the Specification section, two conceptual models have been defined because of the changes in the planning, two different designs of the Data Management Layer will be presented, one for the first phase of the planning and the other for the third one. The following picture shows the Entity-Relationship diagram obtained for the 1st and 3rd phases of the planning.

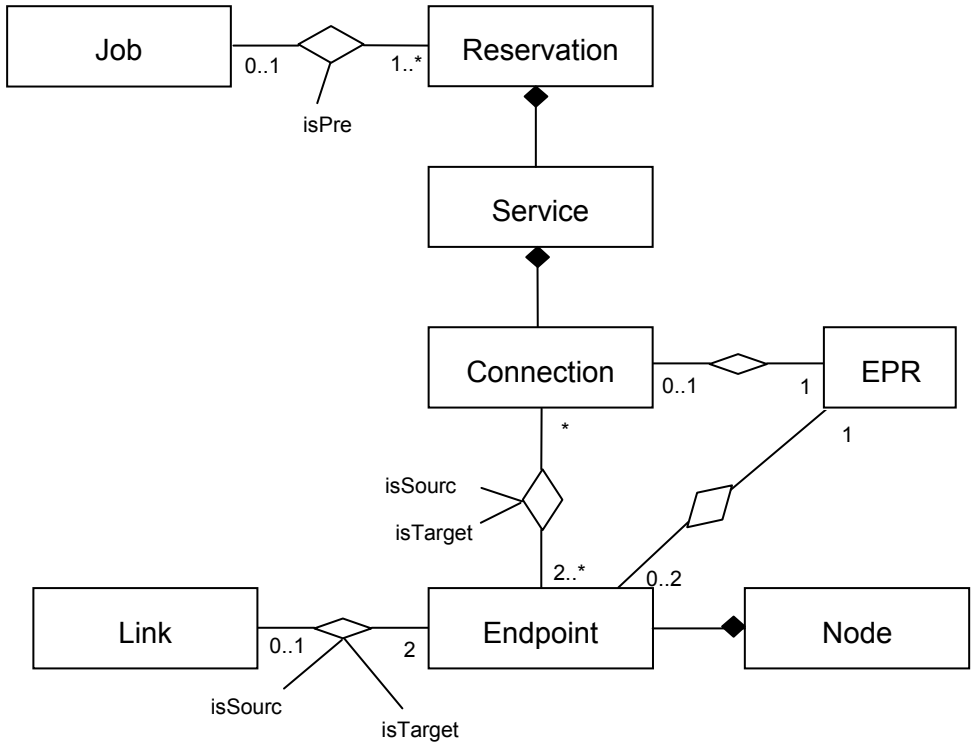


Figure 72. Conceptual Design 1st and 2nd phase

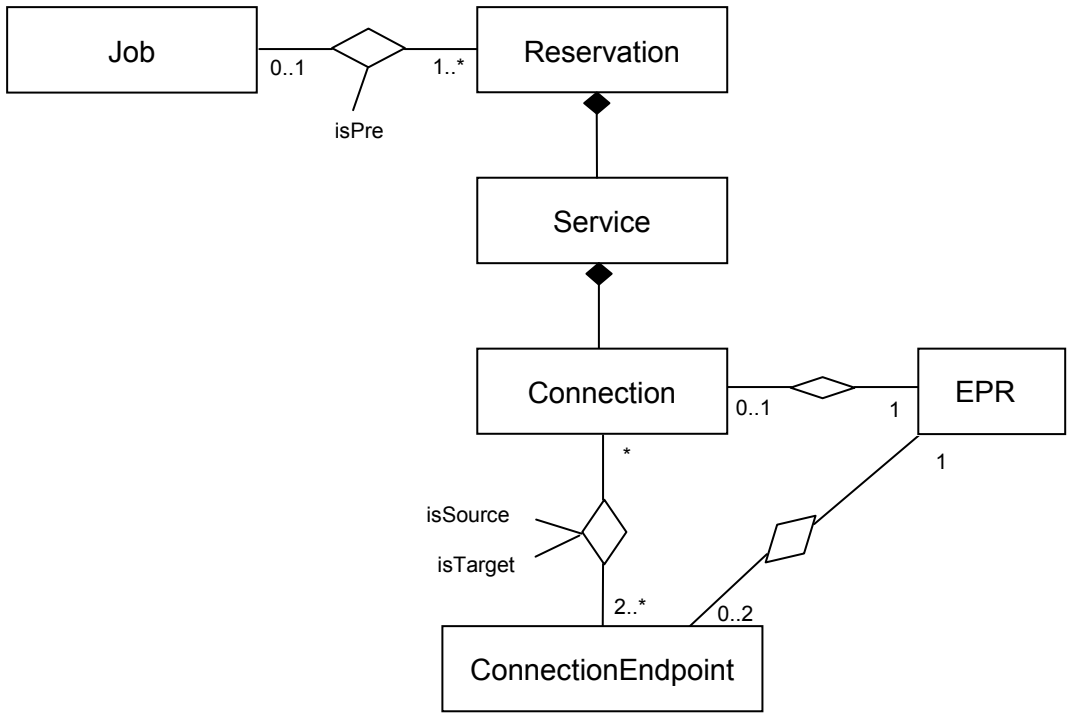


Figure 73. Conceptual Design 3rd phase

8.3.2. Logical Design of the Database

From the conceptual design a translation into a SQL tables is done. The definition of these tables is the logical design of the database.

Primary key

Foreign Key

Primary key + Foreign key

The following table contains the definition of the tables of the database corresponding to the first picture of the conceptual design (1st phase of the planning).

Name of the table	Attributes
Job	jobID
Reservation	reservationID, isPre, *jobID* - jobID is foreign key of Job
Service	serviceID, *reservationID*, typeOfReservation, automaticActivation, expectedStatusChange, status - reservationID is foreign key of Reservation
Fixed Reservation	startTime, duration, *serviceID* - serviceID is foreign key of Service
Deferrable Reservation	earliestStartTime, duration, deadline, earliestStartTimeDef, *serviceID* - serviceID is foreign key of Service
Malleable Reservation	earliestStartTime, amountOfData, deadline, earliestStartTimeDef, durationDef, bwDef, *serviceID* - serviceID is foreign key of Service
Connection	connectionID, *serviceID*, *reservationID*, directionality, actualBW, minBW, maxBW, maxDelay, status, *epr* - serviceID is foreign key of Service - reservationID is foreign key of Reservation - epr is foreign key of EPR
Connection Endpoint	*connectionID*, *serviceID*, *reservationID*, *endpointID*, *epr*, isSource, isTarget - connectionID is foreign key of Connection - serviceID is foreign key of Service - reservationID is foreign key of Reservation - endpointID is foreign key of Endpoint - epr is foreign key of Endpoint
Endpoint	endpointID, *nodeID*, *linkID*, *epr*, name, bw, IPaddress, type, description, inUse - epr is foreign key of EPR

Node	nodeID, name, networkAddress
Link	linkID, name, bw, delay
EPRWrapper	keyValue, keyName, KeyNameSpace, instanceUri

Figure 74. Tables of the Database 1st phase

The following table contains the definition of the tables of the database corresponding to the second picture of the conceptual design (3rd phase of the planning).

Name of the table	Attributes
Job	jobID
Reservation	reservationID, isPre, *jobID* - jobID is foreign key of Job
Service	serviceID, *reservationID*, typeOfReservation, automaticActivation, expectedStatusChange, status - reservationID is foreign key of Reservation
Fixed Reservation	startTime, duration, *serviceID* - serviceID is foreign key of Service
Deferrable Reservation	earliestStartTime, duration, deadline, earliestStartTimeDef, *serviceID* - serviceID is foreign key of Service
Malleable Reservation	earliestStartTime, amountOfData, deadline, earliestStartTimeDef, durationDef, bwDef, *serviceID* - serviceID is foreign key of Service
Connection	connectionID, *serviceID*, *reservationID*, directionality, actualBW, minBW, maxBW, maxDelay, status, *epr* - serviceID is foreign key of Service - reservationID is foreign key of Reservation - epr is foreign key of EPR
Connection Endpoint	*connectionID*, *serviceID*, *reservationID*, *endpointID*, *epr*, isSource, isTarget - connectionID is foreign key of Connection - serviceID is foreign key of Service - reservationID is foreign key of Reservation - endpointID is foreign key of Endpoint - epr is foreign key of Endpoint
EPRWrapper	keyValue, keyName, KeyNameSpace, instanceUri

Table 7. Tables of the Database 2nd phase

9. IMPLEMENTATION

Once done the design of the functionalities and the design of the database, the implementation of the system can be done.

9.1. DOMAIN LAYER

As it has been mention in the last section, the communication between the users and the domain layer is done by means of an interface. This interface specifies which functionalities can the user invoke and which are the input and output parameters. Following two files can be seen: the first one (Reservation.wsdl) specifies the functionalities and its input and output parameters; the second one (Reservation-Types.xsd) specifies the data types for the corresponding input and output parameters.

Not all the functionalities and data types are shown, because the documentation would lengthen unnecessarily.

Reservation.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="Reservation"
      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:resv="http://www.uclp.ca/services/2007/05/reservation"
      xmlns:tns="http://www.uclp.ca/services/2007/05/reservation"
      targetNamespace="http://www.uclp.ca/services/2007/05/reservati
on">

      <wsdl:types>
            <xs:schema>
                  <xs:import
namespace="http://www.uclp.ca/services/2007/05/reservation"
schemaLocation="Reservation-Types.xsd"/>
            </xs:schema>
      </wsdl:types>

      <!-- MESSAGES -->

      <wsdl:message name="NetworkReservationFault">
          <wsdl:part name="NetworkReservationException"
                  element="resv:NetworkReservationException"/>
      </wsdl:message>

      <wsdl:message name="getEndpoints">
            <wsdl:part name="getEndpoints"
                  element="resv:getEndpoints"/>
      </wsdl:message>

      <wsdl:message name="getEndpointsResponse">
            <wsdl:part name="getEndpointsResponse"
                  element="resv:getEndpointsResponse"/>
      </wsdl:message>

      <wsdl:message name="getFeatures">
          <wsdl:part name="getFeatures" element="resv:getFeatures"/>
      </wsdl:message>

      <wsdl:message name="getFeaturesResponse">
            <wsdl:part name="getFeaturesResponse"
```

```
                  element="resv:getFeaturesResponse"/>
      </wsdl:message>
      [...]

      <wsdl:message name="isAvailable">
            <wsdl:part name="isAvailable"
                  element="resv:isAvailable"/>
      </wsdl:message>

      <wsdl:message name="isAvailableResponse">
            <wsdl:part name="isAvailableResponse"
            element="resv:isAvailableResponse"/>
      </wsdl:message>

      <wsdl:message name="createReservation">
            <wsdl:part name="createReservation"
                  element="resv:createReservation"/>
      </wsdl:message>

      <wsdl:message name="createReservationResponse">
            <wsdl:part name="createReservationResponse"
                  element="resv:createReservationResponse"/>
      </wsdl:message>
      [...]

      <!-- PORT TYPE -->
      <wsdl:portType name="ReservationPortType">
            <wsdl:operation name="getEndpoints">
                  <wsdl:input message="resv:getEndpoints"/>
                  <wsdl:output message="resv:getEndpointsResponse"
/>
                  <wsdl:fault name="NetworkReservationFault"
message="resv:NetworkReservationFault"/>
            </wsdl:operation>

            <wsdl:operation name="getFeatures">
                  <wsdl:input message="resv:getFeatures" />
                  <wsdl:output message="resv:getFeaturesResponse"/>
                  <wsdl:fault name="NetworkReservationFault"
message="resv:NetworkReservationFault"/>
            </wsdl:operation>
            [...]

            <wsdl:operation name="isAvailable">
                  <wsdl:input message="resv:isAvailable"/>
                  <wsdl:output message="resv:isAvailableResponse"/>
                  <wsdl:fault name="NetworkReservationFault"
message="resv:NetworkReservationFault"/>
            </wsdl:operation>

            <wsdl:operation name="createReservation">
                  <wsdl:input message="resv:createReservation"/>
                  <wsdl:output
message="resv:createReservationResponse"/>
                  <wsdl:fault name="NetworkReservationFault"
message="resv:NetworkReservationFault"/>
            </wsdl:operation>

            [...]
```

Reservation-Types.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
xmlns:resv="http://www.uclp.ca/services/2007/05/reservation"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
attributeFormDefault="unqualified" elementFormDefault="qualified"
targetNamespace="http://www.uclp.ca/services/2007/05/reservation">

      <xs:complexType name="AAAUserType">
            <xs:annotation>
                  <xs:documentation>Contains AAA information of the
user</xs:documentation>
            </xs:annotation>
            <xs:sequence>
                  <xs:element minOccurs="0" name="UserID"
type="xs:string">
                        <xs:annotation>
                              <xs:documentation>The identity by which
the user is known to the home organization. Optional in case the
Authentication Method field is set to None.</xs:documentation>
                        </xs:annotation>
                  </xs:element>
                  <xs:element name="Secret" type="xs:string">
                        <xs:annotation>
                              <xs:documentation>The users proof of
identity, it could be either a password or a certificate. It must be
set to NONE in case AuthenticationMethod = NONE.</xs:documentation>
                        </xs:annotation>
                  </xs:element>
            </xs:sequence>
      </xs:complexType>
      [...]

      <xs:complexType name="EndpointType">
            <xs:annotation>
                  <xs:documentation>
                        Information about an endpoint
                  </xs:documentation>
            </xs:annotation>
            <xs:sequence>
                  <xs:element name="EndpointId"
                        type="resv:EndpointIdentifierType">
                        <xs:annotation>
      [...]

      <!-- Elements _____________________________________________-->
      <!-- IsAvailable__________________________________________-->
      <xs:element name="isAvailable">
            <xs:annotation>
                  <xs:documentation>
                        Checks the availability of network resources
                  </xs:documentation>
            </xs:annotation>
            <xs:complexType>
                  <xs:sequence>
                        <xs:element name="isAvailable"
                              type="resv:IsAvailableType"/>
                  </xs:sequence>
```

```
            </xs:complexType>
      </xs:element>
      <xs:element name="isAvailableResponse">
            <xs:annotation>
                  <xs:documentation>
                        Answers an Availability Req message
                  </xs:documentation>
            </xs:annotation>
            <xs:complexType>
                  <xs:sequence>
                        <xs:element name="isAvailableResponse"
                              type="resv:IsAvailableResponseType"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <!-- CreateReservation ____________________________________-->
      <xs:element name="createReservation">
            <xs:annotation>
                  <xs:documentation>
                        Reserve network resources
                  </xs:documentation>
            </xs:annotation>
            <xs:complexType>
                  <xs:sequence>
                        <xs:element name="createReservation"
                              type="resv:CreateReservationType"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="createReservationResponse">
            <xs:annotation>
                  <xs:documentation>
                        Answers a Reservation Req message
                  </xs:documentation>
            </xs:annotation>
            <xs:complexType>
                  <xs:sequence>
                        <xs:element name="createReservationResponse"
                        type="resv:CreateReservationResponseType"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <!-- GetStatus ____________________________________________-->
      <xs:element name="getStatus">
            <xs:annotation>
                  <xs:documentation>
                        Retrieves the status of a reservation
                  </xs:documentation>
            </xs:annotation>
            <xs:complexType>
                  <xs:sequence>
                        <xs:element name="getStatus"
                              type="resv:GetStatusType"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="getStatusResponse">
            <xs:annotation>
                  <xs:documentation>
                        Answers a reservation Status Req message
```

```
                  </xs:documentation>
            </xs:annotation>
            <xs:complexType>
                  <xs:sequence>
                        <xs:element name="getStatusResponse"
                              type="resv:GetStatusResponseType"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      [...]

<!-- Identifier-Types _____________________________________________-->
      <xs:simpleType name="ConnectionIdentifierType">
            <xs:annotation>
                  <xs:documentation>
                        Type used for connection IDs (unique within
                        a single service)
                  </xs:documentation>
            </xs:annotation>
            <xs:restriction base="xs:int"/>
      </xs:simpleType>
      <xs:simpleType name="ServiceIdentifierType">
            <xs:annotation>
                  <xs:documentation>
                        Type used for service IDs (unique within a
                        Single reservation)
                  </xs:documentation>
            </xs:annotation>
            <xs:restriction base="xs:int"/>
      </xs:simpleType>
      <xs:simpleType name="ReservationIdentifierType">
            <xs:annotation>
                  <xs:documentation>
                        Type used for reservation IDs
                  </xs:documentation>
            </xs:annotation>
            <xs:restriction base="xs:long"/>
      </xs:simpleType>
      <xs:simpleType name="JobIdentifierType">
            <xs:annotation>
                  <xs:documentation>Type used for job
IDs</xs:documentation>
            </xs:annotation>
            <xs:restriction base="xs:long"/>
      </xs:simpleType>
      <xs:simpleType name="StatusType">
            <xs:restriction base="xs:string">
                  <xs:enumeration value="unknown"/>
                  <xs:enumeration value="pending"/>
                  <xs:enumeration value="active"/>
                  <xs:enumeration value="completed"/>
                  <xs:enumeration value="cancelled_by_user"/>
                  <xs:enumeration value="cancelled_by_system"/>
                  <xs:enumeration value="setup_in_progress"/>
                  <xs:enumeration value="teardown_in_progress"/>
            </xs:restriction>
      </xs:simpleType>
      <xs:simpleType name="ReservationType">
            <xs:annotation>
```

```
                <xs:documentation>
                  Type of reservation.
                </xs:documentation>
            </xs:annotation>
            <xs:restriction base="xs:string">
                  <xs:enumeration value="fixed"/>
                  <xs:enumeration value="deferrable"/>
                  <xs:enumeration value="malleable"/>
            </xs:restriction>
      </xs:simpleType>
      <!-- predefined Types _____________________________________-->
      <xs:complexType name="ConnectionType">
            <xs:annotation>
                  <xs:documentation>Connection
                        type</xs:documentation>
            </xs:annotation>
            <xs:sequence>
                  <xs:element name="ConnectionID"
                        type="resv:ConnectionIdentifierType">
                        <xs:annotation>
                              <xs:documentation>
                                    Identifier of the connection,
                                    unique within a single service
                              </xs:documentation>
                        </xs:annotation>
                  </xs:element>
                  <xs:element name="Source"
                        type="resv:EndpointType">
                        <xs:annotation>
                              <xs:documentation>
                                    Source end point TNA
                              </xs:documentation>
                        </xs:annotation>
                  </xs:element>
                  <xs:element name="Target" type="resv:EndpointType"
                        maxOccurs="unbounded">
                        <xs:annotation>
                              <xs:documentation>
                                    Destination(s) end point(s)
                                    TNA(s)
                              </xs:documentation>
                        </xs:annotation>
                  </xs:element>
                  <xs:element name="Directionality" type="xs:int">
                        <xs:annotation>
                              <xs:documentation>
                                    Possible values:
                                    0="unidirectional tree",
                                    1="bidirectional tree", 3="full
                                    mesh"
                              </xs:documentation>
                        </xs:annotation>
                  </xs:element>
            </xs:sequence>
      </xs:complexType>
      <xs:complexType name="ConnectionConstraintType">
            <xs:annotation>
                  <xs:documentation>
                        Connection Constraint type
                  </xs:documentation>
            </xs:annotation>
```

```
            <xs:complexContent>
                  <xs:extension base="resv:ConnectionType">
                        <xs:sequence>
                              <xs:element name="MinBW"
                                    type="xs:int">
                                    <xs:annotation>
                                          <xs:documentation>
                                                Minimum bandwidth in
                                                Mbps
                                          </xs:documentation>
                                    </xs:annotation>
                              </xs:element>
                                    <xs:element name="MaxBW"
                                    type="xs:int" minOccurs="0">
                                    <xs:annotation>
                                          <xs:documentation>
                                                Maximum bandwidth in
                                                Mbps
                                          </xs:documentation>
                                    </xs:annotation>
                              </xs:element>
                              <xs:element name="MaxDelay"
                              type="xs:int" minOccurs="0">
                                    <xs:annotation>
                                          <xs:documentation>
                                                Maximum delay in
                                                milliseconds
                                          </xs:documentation>
                                    </xs:annotation>
                              </xs:element>
                              <xs:element name="DataAmount"
                              type="xs:long" minOccurs="0">
                                    <xs:annotation>
                                          <xs:documentation>
                                    Indicates the amount of data to
                                    be transferred,
                                    in MBytes
                                    (required for and only for malleable
                                    reservations!)</xs:documentation>
                                    </xs:annotation>
                              </xs:element>
                        </xs:sequence>
                  </xs:extension>
            </xs:complexContent>
      </xs:complexType>
      <xs:complexType name="ConnectionStatusType">
            <xs:annotation>
                  <xs:documentation>Connection status
                  type</xs:documentation>
            </xs:annotation>
            <xs:complexContent>
                  <xs:extension base="resv:ConnectionType">
                        <xs:sequence>
                              <xs:element name="Status"
                              type="resv:StatusType"/>
                              <xs:element name="DomainStatus"
type="resv:DomainStatusType" minOccurs="0" maxOccurs="unbounded"/>
                              <xs:element name="ActualBW"
                              type="xs:int">
                                    <xs:annotation>
                                          <xs:documentation>
```

```
                                                    Actual bandwidth in
                                                    Mbps
                                                </xs:documentation>
                                            </xs:annotation>
                                        </xs:element>
                                    </xs:sequence>
                                </xs:extension>
                            </xs:complexContent>
                        </xs:complexType>
                        [...]
```

9.2. DATABASE MANAGEMENT LAYER

For the implementation of the database management layer, the 'Data Mapper' pattern is used. The 'Data Mapper' is a pattern of persistence in the relational databases that manages the correspondence between the objects and the database. In this pattern, the designer defines the correspondence between the classes diagram and the relational schema. In this project the Data Mapper used is Hibernate.

Hibernate is the implementation of the design pattern Data Mapper that isolates the objects model from the database. To design the persistence layer, the following steps must be done:

- design the relational database
- define the persistent classes
- define the correspondence between the classes structure and the tables structure (the mappings, see below)
- configure the connection method and the SQL dialect (Hibernate properties, see below)

9.2.1. Database

The definition of the database can be seen in the section 8.3.

9.2.2. Persistent Classes

The persistent classes to be mapped are the following ones defined in the database. Just remember the names:

- Job
- Reservation
- Service
- FixedReservation
- DeferrableReservation
- MalleableReservation
- Connection
- ConnectionEndpoint
- EPR

9.2.3. Mappings

Job

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-
3.0.dtd">
<hibernate-mapping>
    <class
      name = "ca.uclp.services.adreservations.impl.dbclasses.Job"
      table = "JOB">
        <id
            name = "jobIDDB"
            column = "JOB_ID_DB"
            unsaved-value = "0">
            <generator class = "increment"/>
        </id>
        <property
            name = "jobID"
            column = "JOB_ID"
            type = "long"
            unique = "true"/>
        <bag
            name = "preReservations"
            inverse = "true"
            lazy = "false">
            <key column = "JOB_ID_DB"/>
            <one-to-many class =
      "ca.uclp.services.adreservations.impl.dbclasses.Reservation"/>
        </bag>
    </class>
</hibernate-mapping>
```

Reservation

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-
3.0.dtd">

<hibernate-mapping>
      <class
      name =
"ca.uclp.services.adreservations.impl.dbclasses.Reservation"
      table = "RESERVATION">

        <id
            name = "reservationIDDB"
            column = "RESERVATION_ID_DB"
            unsaved-value = "0">
            <generator class = "increment"/>
        </id>

        <property
            name = "reservationID"
            column = "RESERVATION_ID"
            type = "long"
            unique = "true"/>
        <property
            name = "isPre"
            column = "IS_PRE"
            type = "boolean"/>
        <property
            name = "notificationConsumerURL"
            column = "NOTIFICATION_CONSUMER_URL"
            type = "string"/>
        <property
            name = "owner"
            column = "OWNER"
            type = "string"/>

        <bag
            name = "services"
            inverse = "true"
            lazy = "false"
            cascade = "all">
            <key column = "RESERVATION_ID_DB"/>
            <one-to-many class =
"ca.uclp.services.adreservations.impl.dbclasses.Service"/>
        </bag>

        <many-to-one
            name = "job"
            column = "JOB_ID_DB"
              class =
"ca.uclp.services.adreservations.impl.dbclasses.Job"
                  not-null = "false"
                  lazy = "false"/>
    </class>
</hibernate-mapping>
```

Service

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-
3.0.dtd">

<hibernate-mapping>

    <class
      name =
"ca.uclp.services.adreservations.impl.dbclasses.Service"
      table = "SERVICE">
        <id
            name = "serviceIDDB"
            column = "SERVICE_ID_DB"
            unsaved-value = "0">
            <generator class = "increment"/>
        </id>

        <property
            name = "serviceID"
            column = "SERVICE_ID"
            type = "integer"/>
        <property
            name = "typeOfReservation"
            column = "TYPE_OF_RESERVATION"
            type = "string"/>
        <property
            name = "expectedStatusChange"
            column = "EXPECTED_STATUS_CHANGE"
            type = "calendar"/>
        <property
            name = "status"
            column = "STATUS"
            type = "string"/>
        <property
            name = "automaticActivation"
            column = "AUTOMATIC_ACTIVATION"
            type = "boolean"/>

        <many-to-one
            name = "reservation"
            column = "RESERVATION_ID_DB"
              class =
"ca.uclp.services.adreservations.impl.dbclasses.Reservation"
                not-null = "true"
                lazy = "false"
                cascade = "all"/>

        <one-to-one
            name = "fixedReservation"
            class =
"ca.uclp.services.adreservations.impl.dbclasses.FixedReservation"
            cascade = "all"/>
        <one-to-one
            name = "deferrableReservation"
            class =
```

```
"ca.uclp.services.adreservations.impl.dbclasses.DeferrableReservatio
n"
            cascade = "all"/>
        <one-to-one
            name = "malleableReservation"
            class =
"ca.uclp.services.adreservations.impl.dbclasses.MalleableReservation
"
            cascade = "all"/>

        <bag
            name = "connections"
            inverse = "true"
            lazy = "false"
            cascade = "all">
            <key column = "SERVICE_ID_DB"/>
            <one-to-many class =
"ca.uclp.services.adreservations.impl.dbclasses.Connection"/>
        </bag>
    </class>
</hibernate-mapping>
```

FixedReservation

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-
3.0.dtd">

<hibernate-mapping>

    <class
      name =
"ca.uclp.services.adreservations.impl.dbclasses.FixedReservation"
      table = "FIXED_RESERVATION">
        <id
            name = "fixedReservationID"
            column = "FIXED_RESERVATION_ID"
            unsaved-value = "0">
            <generator class = "increment"/>
        </id>

        <property
            name = "startTime"
            column = "START_TIME"
            type = "long"/>
        <property
            name = "duration"
            column = "DURATION"
            type = "integer"/>
    </class>
</hibernate-mapping>
```

DeferrableReservation

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-
3.0.dtd">

<hibernate-mapping>

    <class
      name =
"ca.uclp.services.adreservations.impl.dbclasses.DeferrableReservatio
n"
      table = "DEFERRABLE_RESERVATION">
        <id
            name = "deferrableReservationID"
            column = "DEFERRABLE_RESERVATION_ID"
            unsaved-value = "0">
            <generator class = "increment"/>
        </id>

        <property
            name = "startTime"
            column = "START_TIME"
            type = "calendar"/>
        <property
            name = "duration"
            column = "DURATION"
            type = "integer"/>
        <property
            name = "deadline"
            column = "DEADLINE"
            type = "calendar"/>
        <property
            name = "startTimeDef"
            column = "START_TIME_DEF"
            type = "calendar"/>
    </class>
</hibernate-mapping>
```

MalleableReservation

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-
3.0.dtd">

<hibernate-mapping>

    <class
      name                                                          =
"ca.uclp.services.adreservations.impl.dbclasses.MalleableReservation
"
      table = "MALLEABLE_RESERVATION">

        <id
```

```
            name = "malleableReservationID"
            column = "MALLEABLE_RESERVATION_ID"
            unsaved-value = "0">
            <generator class = "increment"/>
        </id>

        <property
            name = "startTime"
            column = "sSTART_TIME"
            type = "calendar"/>
        <property
            name = "deadline"
            column = "DEADLINE"
            type = "calendar"/>
        <property
            name = "startTimeDef"
            column = "START_TIME_DEF"
            type = "calendar"/>
        <property
            name = "durationDef"
            column = "DURATION_DEF"
            type = "integer"/>
        <property
            name = "bwDef"
            column = "BW_DEF"
            type = "integer"/>
    </class>
</hibernate-mapping>
```

Connection

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-
3.0.dtd">

<hibernate-mapping>

    <class
      name =
"ca.uclp.services.adreservations.impl.dbclasses.Connection"
      table = "CONNECTION">
        <id
            name = "connectionIDDB"
            column = "CONNECTION_ID_DB"
            unsaved-value = "0">
            <generator class = "increment"/>
        </id>

        <property
            name = "connectionID"
            column = "CONNECTION_ID"
            type = "integer"/>
        <property
            name = "minBW"
            column = "MIN_BW"
            type = "integer"/>
```

```
        <property
            name = "maxBW"
            column = "MAX_BW"
            type = "integer"/>
        <property
            name = "maxDelay"
            column = "MAX_DELAY"
            type = "integer"/>
        <property
            name = "directionality"
            column = "DIRECTIONALITY"
            type = "integer"/>
        <property
            name = "actualBW"
            column = "ACTUAL_BW"
            type = "integer"/>
        <property
            name = "status"
            column = "STATUS"
            type = "string"/>
        <property
            name = "dataAmount"
            column = "DATA_AMOUNT"
            type = "long"/>

        <many-to-one
            name = "epr"
            column = "EPR"
              class =
"ca.uclp.uclpv2.gui.editors.common.model.EPRWrapper"
                  not-null = "false"
                  lazy = "false"
                  cascade = "all"/>

        <many-to-one
            name = "service"
            column = "SERVICE_ID_DB"
              class =
"ca.uclp.services.adreservations.impl.dbclasses.Service"
                  not-null = "true"
                  lazy = "false"
                  cascade = "all"/>

        <bag
            name = "endpoints"
            inverse = "true"
            lazy = "false"
            cascade = "all">
            <key column = "CONNECTION_ID_DB"/>
            <one-to-many class =
"ca.uclp.services.adreservations.impl.dbclasses.ConnectionEndpoint"/
>
        </bag>
    </class>
</hibernate-mapping>
```

ConnectionEndpoint

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-
3.0.dtd">

<hibernate-mapping>

    <class
      name =
"ca.uclp.services.adreservations.impl.dbclasses.ConnectionEndpoint"
      table = "CONNECTION_ENDPOINT">
        <id
            name = "connectionEndpointID"
            column = "CONNECTION_ENDPOINT_ID"
            unsaved-value = "0">
            <generator class = "increment"/>
        </id>

        <property
            name = "endpointID"
            column = "ENDPOINT_ID"
            type = "integer"/>
        <property
            name = "isSource"
            column = "IS_SOURCE"
            type = "boolean"/>
        <property
            name = "isTarget"
            column = "IS_TARGET"
            type = "boolean"/>
        <property
            name = "keyValueEPR"
            column = "KEY_VALUE_EPR"
            type = "string"/>

            <many-to-one
            name = "connection"
            column = "CONNECTION_ID_DB"
              class =
"ca.uclp.services.adreservations.impl.dbclasses.Connection"
                  cascade = "all"
                  lazy = "false" />
    </class>
</hibernate-mapping>
```

EPRWrapper

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-
3.0.dtd">

<hibernate-mapping>

    <class
      name = "ca.uclp.uclpv2.gui.editors.common.model.EPRWrapper"
      table = "EPR_WRAPPER">
        <id
            name = "keyId"
            column = "KEY_ID"
            unsaved-value = "0">
            <generator class = "increment"/>
        </id>

        <property
            name = "keyValue"
            column = "KEY_VALUE"
            type = "string"
            unique = "true"/>
        <property
            name = "keyName"
            column = "KEY_NAME"
            type = "string"/>
        <property
            name = "instanceURI"
            column = "INSTANCE_URI"
            type = "string"/>
        <property
            name = "keyNamespace"
            column = "KEY_NAMESPACE"
            type = "string"/>
    </class>
</hibernate-mapping>
```

9.2.4. Hibernate Properties

To configure the connection method and the SQL dialect, a file called 'hibernate.properties' is created:

```
hibernate.connection.driver_class = org.postgresql.Driver
hibernate.connection.url =
 jdbc:postgresql://localhost:5432/uclpdb
hibernate.connection.username = postgres
hibernate.connection.password = postgres
hibernate.connection.pool_size = 10
hibernate.connection.autocommit = true
hibernate.current_session_context_class = thread
```

```
hibernate.hbm2ddl.auto = update
hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
hibernate.c3p0.acquire_increment=1
hibernate.c3p0.idle_test_period=100
hibernate.c3p0.min_size=10
hibernate.c3p0.max_size=100
hibernate.c3p0.timeout=100
```

In this file the following information is given:

- The URL of the database: jdbc:postgresql://localhost:5432/uclpdb
- The dialect: PosgreSQL
- The user name and the password of the database (in this case 'password' in both ones)
- And other features of the system.

9.3. TOOLS USED

In this section all the software used to implement the project presented in this documentation is going to be described.

First of all, the language used to write all the code of the system is the Java version 1.5.0. This language has been used because permits the product to be independent from the Operating System and it is free software.

To continue, the tools used to implement the database are going to be explained. The ORDBMS (object-relational database management system) used is PostgreSQL [13]. At the beginning of the project the Database Management System MySQL was used, but later it was seen that MySQL was too simple for some needs of the project. So, it was decided to use the PostgreSQL, which is an open source product and also has a free driver, JDBC (Java DataBase Connector), which permits the communication between applications developed in Java. As it has been explained in the last section, to map the classes of the Reservation Management group data structure into the tables of the database, the ORM (object-relational mapping) middleware Hibernate [14] is used (see the mappings in the section 9.2.3).

As the ARS is a Web Service of the UCLP system, it is needed a technology for building the web service component and to allow the UCLP system to share its resources with the ARS. So, in this implementation it is used the Globus Toolkit [15]. This technology permits the ARS to expose a web service interface and to communicate to the UCLP system in order to read the topology of the network and to create or delete lightpaths to create a new connection or delete an existent one. To identify these lightpaths, an "Endpoint Reference" is used. The endpoint reference is a data type that contains four strings: keyValue (the identifier, it is unique), keyName, keyNameSpace and instanceURI (see section 9.2.3). KeyValue is used as the identifier of a Lightpath in the communication between the ARS and the UCLP system.

If a connection has to be activated automatically, a scheduler is needed. When a reservation is created, the scheduler is programmed at the requested start time. When the requested start time arrives, the lightpaths needed for the connection are created physically calling the UCLP system. In the same way, when the end time arrives, the

UCLP system is called to delete the lightpaths. To implement this scheduler the Quartz [16] library is used.

Finally, to implement the “simple” GUI to test the service implemented, a web application has been developed. The web server used has been the Tomcat 6.0 because it is free software and is also Operating System independent. JSP’s (Java Server Pages) and Servlet’s have been used to develop this web application because they are implemented in Java, the same technology used to generate the source code of the system.

10. TESTING

Testing is a definitive procedure that produces a test result. It helps to find errors in the system, not to determine if the product works well or not.

10.1. TEST PLAN

For each use case described in section 4.4.2, it has to be proved that it works properly. To test its correctness, the tests to make and the output of the functionality if the tests are correct are described. Firstly, the correctness of the input parameters must be checked (to see the inputs and outputs of each use case, see the section 8.2). Next, the 'tests to make' describe the different cases that should give a controlled error.

10.1.1. Get Features

It must be checked that the user is valid; it has to be authenticated.

Tests to make

Cases that should give a controlled error:

1. The user is not authorized.

Output

If the user is valid, a list of the features of the system is returned.

10.1.2. Get Endpoints

It must be checked that the user is valid, it has to be authenticated.

Tests to make

Cases that should give a controlled error:

1. The user is not authorized.

Output

If the user is valid, a list of the endpoints of the net is returned.

10.1.3. Get Reservations

It must be checked that the user is valid, it has to be authenticated.

Tests to make

Cases that should give a controlled error:

1. The user is not authorized.

Output

If the user is valid, a list of the reservations done is returned.

10.1.4. Get Jobs

It must be checked that the user is valid, it has to be authenticated.

Tests to make

Cases that should give a controlled error:

1. The user is not authorized.

Output

If the user is valid, a list of the reservations done is returned.

10.1.5. Is Available

In this case the following parameters must be checked:

- For each service requested, it has to be verified its correctness request. The serviceID must be unique in the reservation. The allowed values for typeOfReservation are 1 (fixed), 2 (deferrable) and 3 (malleable). For each connection of the service, the connectionID has to be unique in the service. The endpoint source must exist and be an external endpoint. The possible values for directionality are 0 (unidirectional tree), 1 (bidirectional tree) and 3 (full mesh).
- If jobID is null, a permanent reservation is requested, if it is zero, a pre-reservation is requested and the job doesn't exist, it has to be created. If it is different from zero, a pre-reservation is requested and the job exists. It has to be checked if this job exists.
- Authenticate the user: check that the user is valid.

Tests to make

Cases that should give a controlled error:

1. The user is not authorized.
2. The identifier of the job is a number different than zero and this job does not exist in the DB.
3. There are serviceID's repeated in the same reservation request.
4. There are connectionID's repeated in the same service of the same reservation.
5. The value of 'typeOfReservation' is different from "fixed", "deferrable", "malleable".
6. The value of 'duration' in a FixedReservation is a negative value.
7. The value of 'startTime' in a FixedReservation is not valid: the date is smaller than the actual date.
8. The source endpoint requested in a connection does not exist or is not valid.
9. The target endpoint requested in a connection does not exist or is not valid.
10. The value directionality in a connection is not an integer value or is different from 0, 1 or 3.
11. The value of maxWB and minBW in a connection are negative values
12. The value of delay in a connection is a negative value
13. One or more connections in the reservation request endpoints that are already reserved at the time requested.

Output

For each connection of each service of the reservation requested, it is calculated if this connection is possible between the endpoints at the time requested.

If all the connections of the reservation can be done, the values of the output parameters are:

- For each connection is returned the identifier of the service, the identifier of the reservation and availability equal to zero.
- alternativeStartTime = null.

If there is at least one connection not possible, the output values are:

- For each connection is returned the identifier of the service, the identifier of the reservation and the corresponding availability.
- alternativeStartTime = if exists, the startTime in which this reservation is possible.

If some input parameters are not valid, or some error occurs during the execution, the output values are:

- detailedResult = null
- alternativeStartTime = null

10.1.6. Create Reservation

In this case the following parameters must be checked:

- For each service requested, it has to be verified its correctness request. The serviceID must be unique in the reservation. The allowed values for typeOfReservation are 1 (fixed), 2 (deferrable) and 3 (malleable). For each connection of the service, the connectionID has to be unique in the service. The endpoint source must exist and be an external endpoint. The possible values for directionality are 0 (unidirectional tree), 1 (bidirectional tree) and 3 (full mesh).
- If jobID is null, a permanent reservation is requested, if it is zero, a pre-reservation is requested and the job doesn't exist, it has to be created. If it is different from zero, a pre-reservation is requested and the job exists. It has to be checked if this job exists.
- Authenticate the user: check that the user is valid.

Tests to make

Cases that should give a controlled error:

1. The user is not authorized.
2. The identifier of the job is a number different than zero and this job does not exist in the DB.
3. There are serviceID's repeated in the same reservation request.
4. There are connectionID's repeated in the same service of the same reservation.
5. The value of 'typeOfReservation' is different from "fixed", "deferrable", "malleable".
6. The value of 'duration' in a FixedReservation is a negative value.

7. The value of ‘startTime’ in a FixedReservation is not valid: the date is smaller than the actual date.
8. The source endpoint requested in a connection does not exist or is not valid.
9. The target endpoint requested in a connection does not exist or is not valid.
10. The value directionality in a connection is not an integer value or is different from 0, 1 or 3.
11. The value of maxWB and minBW in a connection are negative values
12. The value of delay in a connection is a negative value
13. One or more connections in the reservation request endpoints that are already reserved at the time requested.

Output

For each connection of each service of the reservation requested, it is calculated if this connection is possible between the endpoints at the time requested.

If all the connections of the reservation can be done, the values of the output parameters are:

- Rid = the identifier of the reservation created.
- Jid = null if the reservation was a permanent reservation; or the identifier of the job if it was a pre-reservation.

If there is at least one connection not possible, or some error occurs during the execution, an exception is thrown.

10.1.7. Cancel Reservation

The following parameters must be checked:

- The identifier of the reservation requested must exist.
- Authenticate the user: check that the user is valid.

Tests to make

Cases that should give a controlled error:

1. The user is not authorized.
2. The identifier of the reservation does not exist in the database.

Output

If the reservation does not exist, FALSE is returned. Otherwise, if the cancellation has been successful, TRUE is returned.

10.1.8. Get Status

The following parameters must be checked:

- The identifier of the reservation requested must exist.
- The identifiers of the services must exist and must belong to the reservation.
- Authenticate the user: check that the user is valid.

Tests to make

Cases that should give a controlled error:

1. The user is not authorized.
2. The identifier of the reservation does not exist in the DB.
3. The identifiers of the services don't exist in the DB or don't belong to the reservation requested.

Output

If the serviceID array is null, the statuses of all the services of the reservation are returned. If it is not null, only the status of the requested services are returned.

If the reservation identifier and the servicesID's given exist in the DB, the status of the servicees and the connections are returned.

If the reservationID is invalid or some serviced does not exist, the detailedStatus array returned is null.

10.1.9. Complete Job

In this case the following parameters must be checked:

- The identifier of the job requested must exist.
- Authenticate the user: check that the user is valid.

Tests to make

Cases that should give a controlled error:

1. The user is not authorized.
2. The identifier of the job does not exist in the database.

Output

If the job does not exist, FALSE is returned. Otherwise, if the job has been completed successfully, TRUE is returned.

10.1.10. Cancel Job

The following parameters must be checked:

- The identifier of the job requested must exist.
- Authenticate the user: check that the user is valid.

Tests to make

Cases that should give a controlled error:

3. The user is not authorized.
4. The identifier of the job does not exist in the database.

Output

If the job does not exist, FALSE is returned. Otherwise, if the job has been cancelled successfully, TRUE is returned.

10.1.11. Activate

The following parameters must be checked:

- The identifier of the reservation requested must exist.

- The identifier of the service must exist and must belong to the reservation.
- Authenticate the user: check that the user is valid.

Tests to make

Cases that should give a controlled error:

5. The user is not authorized.
6. The identifier of the reservation does not exist in the database.
7. The identifier of the service does not exist in the database or does not belong to the reservation.

Output

If the reservation or the service do not exist, FALSE is returned. Otherwise, if the activation has been successfully, TRUE is returned.

10.1.12. Activate Automatically

In this case the following parameters must be checked:

- The identifier of the reservation requested must exist.
- The identifier of the service must exist and must belong to the reservation.

Tests to make

Cases that should give a controlled error:

1. The identifier of the reservation does not exist in the database.
2. The identifier of the service does not exist in the database or does not belong to the reservation.

Output

If the reservation or the service do not exist, FALSE is returned. Otherwise, if the activation has been successfully, TRUE is returned.

10.1.13. Complete Service

The following parameters must be checked:

- The identifier of the reservation requested must exist.
- The identifier of the service must exist and must belong to the reservation.

Tests to make

Cases that should give a controlled error:

1. The identifier of the reservation does not exist in the database.
2. The identifier of the service does not exist in the database or does not belong to the reservation.

Output

If the reservation or the service do not exist, FALSE is returned. Otherwise, if the service has been completed successfully, TRUE is returned.

10.1.14. Reconfigure System

In this use case, there are nor input neither output parameters. It must be checked if the network has been checked and recalculate the paths of the connections if necessary. If some error occurs during the process, an exception is thrown.

10.2. JUnit Test Cases

JUnit is a Java framework for performing unit tests on code. By testing code after every change, programmers can be reassured that changing a small amount of code does not break the larger system. Without automated testing tools like JUnit, retesting can be a tedious and inaccurate process. By allowing the testing process to occur frequently and automatically, you can keep software coding errors at a minimum.

Creating the test cases, however, can be a difficult process requiring knowledge of exactly what the code does and how it should perform. The unit tests performed by JUnit must be carefully created to ensure that the tests rigorously check the code just as it would run in the real world.

Test cases for JUnit are written as Java classes that extend the JUnit framework. These classes have a number of methods, each of which tests a particular function, or *unit*, of the code. Generally, the more test cases written, the more thorough the test will be overall. Every functional area of a product should be tested: Only by doing this the tests will be of real value in making sure that small changes in one place do not have larger implications in another.

For each Use Case described in the section before, a test case has been written and tested for each 'Tests to make' described for each use case. Following there are some test cases shown as an example. Not all the test cases are presented because it would lengthen this documentation unnecessarily.

AdvanceReservationsTest.java

```
[...]

//Use case: Get Endpoints
//The user is not valid
@Test public void testGetEndpointsInvalidUser() throws Exception
{
	String userID = "p";
	String secret = "p";

	logger.info("");
	logger.info(":::::::: GET ENDPOINTS ::::::");
	logger.info(" -> invalid user");

	GetEndpoints reReq = new GetEndpoints();
	GetEndpointsType reReqT = new GetEndpointsType();
	reReqT.setDomain("My domain");
	reReq.setGetEndpoints(reReqT);
	GetEndpointsResponse reRep =
		AdvReservationsWrapper.getEndpoints(reReq, host, port);

	printEndpoints(reRep.getGetEndpointsResponse().getEndpoints()
);
}
```

```
//Use Case: Get Endpoints
//The user is valid
@Test public void testGetEndpointsValidUser() throws Exception
{
	String userID = "laia";
	String secret = "uclp";

	logger.info("");
	logger.info(":::::::: GET ENDPOINTS ::::::");
	logger.info(" -> valid user");

	GetEndpoints reReq = new GetEndpoints();
	GetEndpointsType reReqT = new GetEndpointsType();
	reReqT.setDomain("My domain");
	reReq.setGetEndpoints(reReqT);
	GetEndpointsResponse reRep =
		AdvReservationsWrapper.getEndpoints(reReq, host, port);

	printEndpoints(reRep.getGetEndpointsResponse().getEndpoints())
;
}


[...]


//Use Case: Get Status
//The service requested does not exist
@Test public void testGetStatusInvalidServicesID() throws Exception
{
	String userID = "laia";
	String secret = "uclp";

	System.out.println("");
	System.out.println(":::::::: GET STATUS ::::::");
	System.out.println(" -> invalid services ID");

	GetStatus sReq = new GetStatus();
	GetStatusType gst = new GetStatusType();
	gst.setReservationID((long)1);
	int[] ss = new int[1];
	ss[0] = 100;
	gst.setServiceID(ss);
	AAAUserType aaa = new AAAUserType();
	aaa.setUserID(userID);
	aaa.setSecret(secret);
	gst.setAAAUserInformation(aaa);
	sReq.setGetStatus(gst);
	GetStatusResponse sResp =
		AdvReservationsWrapper.getStatus(sReq, host, port);
	printResultServiceStatus(sResp);
}


[...]
```

10.3. ADVANCE RESERVATION SERVICE DEPLOYMENT

The ARS is deployed in part of i2CAT's experimental optical network. This network is composed by several Nortel OPTera Metro 5200 platforms, a DWDM Optical Add and Drop Multiplexer. The ARS allows the users of the network to reserve in advance Gigabit Ethernet connections over DWDM links.

By the end of October, the ARS was deployed over part of CANARIE network, the Canadian National Research Network. At present, the ARS controls a subset of CANARIE's resources, the ones allocated to the PHOSPHORUS testbed. CANARIE network is made up of three different SONET platforms: the Cisco ONS 15454 platform, the Nortel Optical Multiservice Edge 6500 platform and the Nortel HDXc platform. Users are able to create advance reservations of Gigabit Ethernet endpoints over SONET links.

10.4. GUI

As explained previously, a "simple" GUI has been developed to test the functionalities of the ARS. This GUI has been used in the first phase of the planning to test only the functionalities of the ARS (without connect to the UCLP system) and it has been also used in the second phase to test the ARS connected to the UCLP system.

This GUI done is a web application implemented using jsp's, servlet's and the web server Tomcat (see section 9.3).

All the use cases of the ARS (see section 4.4.2) will be shown except the reconfigureSystem one, because the changes done in the network are not visible in this GUI.

In the following picture, the net used to make the example of this test is shown. The small blue arrows are the external endpoints; the black boxes are the nodes; the discontinuous blue arrows are the links and in the extremes of each links there are the internal endpoints.

Figure 75. An example of network

The following table contains the relation of the endpoints and nodes, in other words which endpoints belong to each node, its IP address and its interface type.

Endpoint ID (external)	Node Name	Node ID	IP address	Interface Type
1	Lleida	4	10.3.1.11	Border
2	Tarragona	5	10.3.1.12	User
3	Barcelona	3	10.3.1.13	Border
4	Girona	2	10.3.1.14	Border
5	Barcelona	3	10.3.1.15	User
6	Lleida	4	10.3.1.16	Border

Table 8. Relation of endpoints and nodes

The table below contains information about which endpoints belong to each link, and the node of the endpoints.

Link ID	Source Node	Endpoint ID (internal)	Destination Node	Endpoint ID (internal)
1	5	7	4	8
2	4	9	1	10
3	1	11	2	12
4	2	13	3	14
5	1	15	3	16
6	4	17	1	18
7	1	19	3	20

Table 9. Relation of links and endpoints

I. Authenticate User

Firstly, the user must be authenticated. The login and password of the user, and the host and the port of the server we wants to connect must be indicated. This is the first page of the web application. On the left, there is a menu with all the available functionalities in this service. To request functionality, the user must just click on the link that has the name of the functionality he wants and follows the required steps (explained in the follow paragraphs).

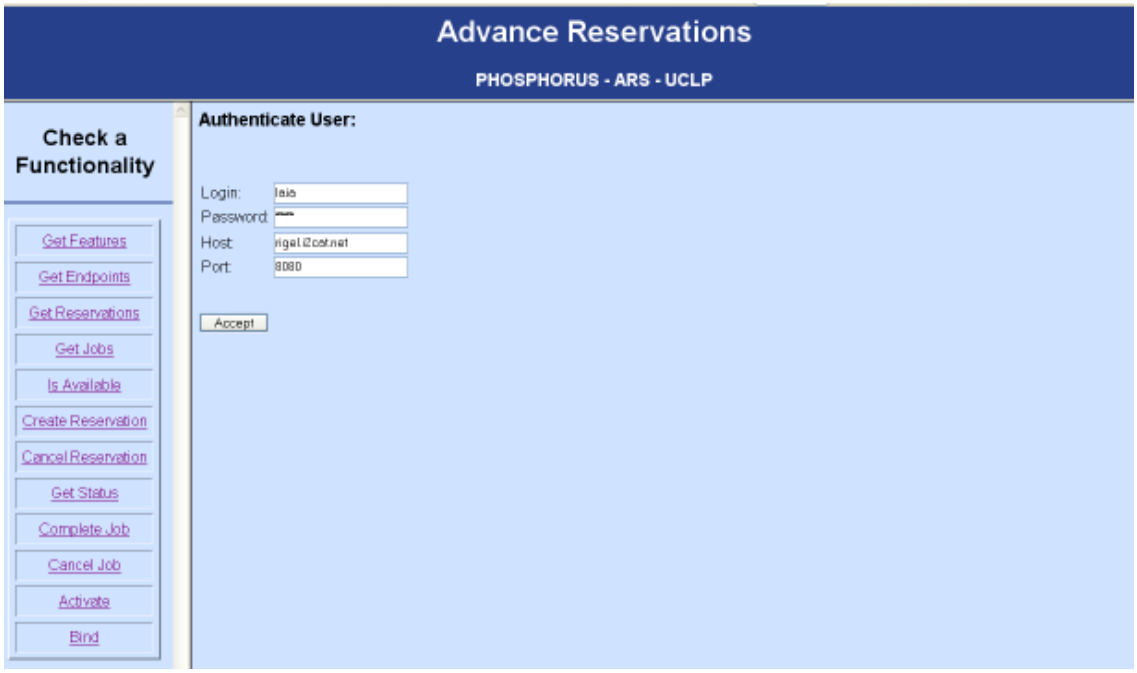


Figure 76. Authenticate User

If the user is available to use the service and the address and port of the host are valid, the user can go on.

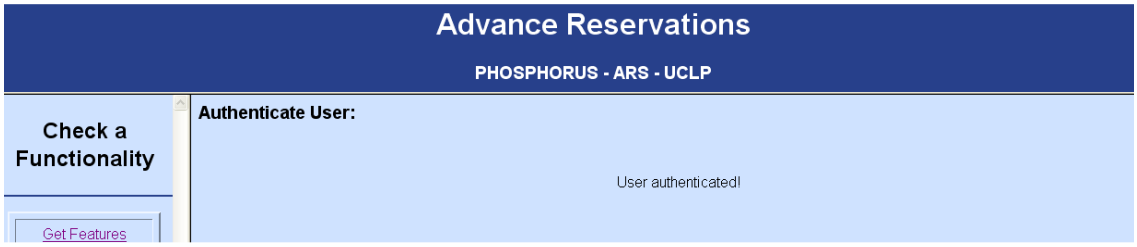


Figure 77. User authenticated

II. Get Features

To get the feature of the system, the user must click on the link on the left 'Get Feature', and after click on the button.

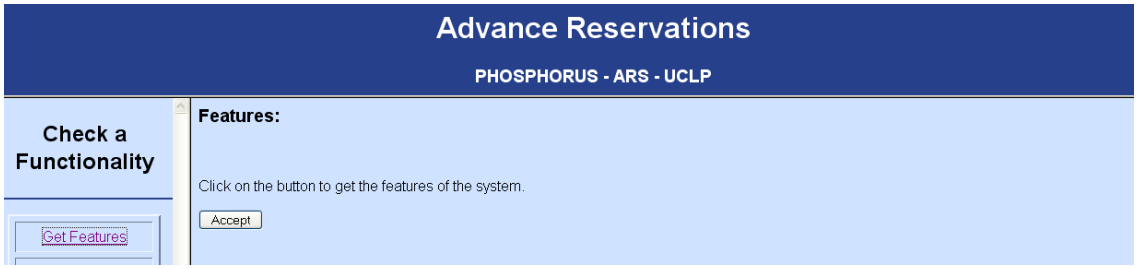


Figure 78. Get Features request

Then, the features of the system are shown.

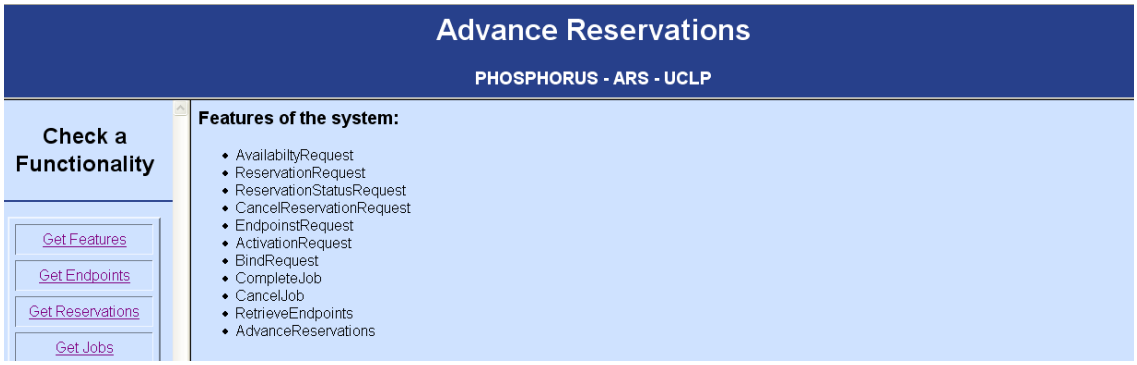


Figure 79. Features of the system

III. Get Endpoints

To get the available endpoints of the network, the user must click on the link 'Get Endpoints' and after click on the button 'Accept'.

Figure 80. Get Endpoints request

Once the button is clicked, the IP addresses of the available endpoints of the network are shown.

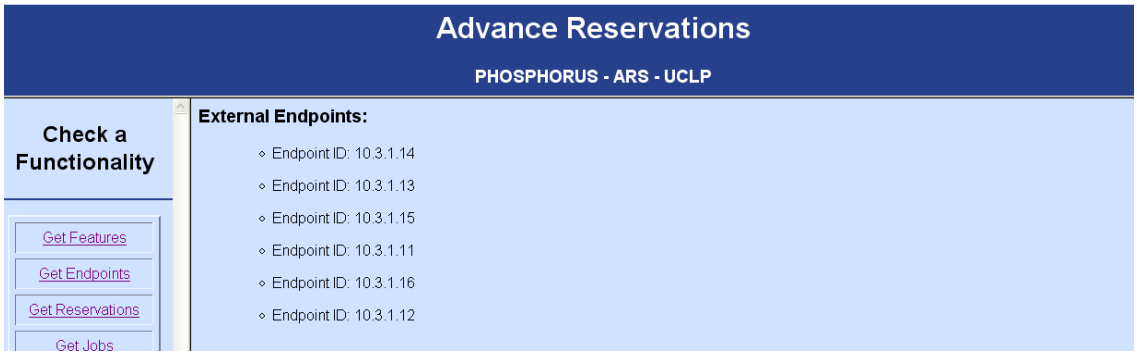
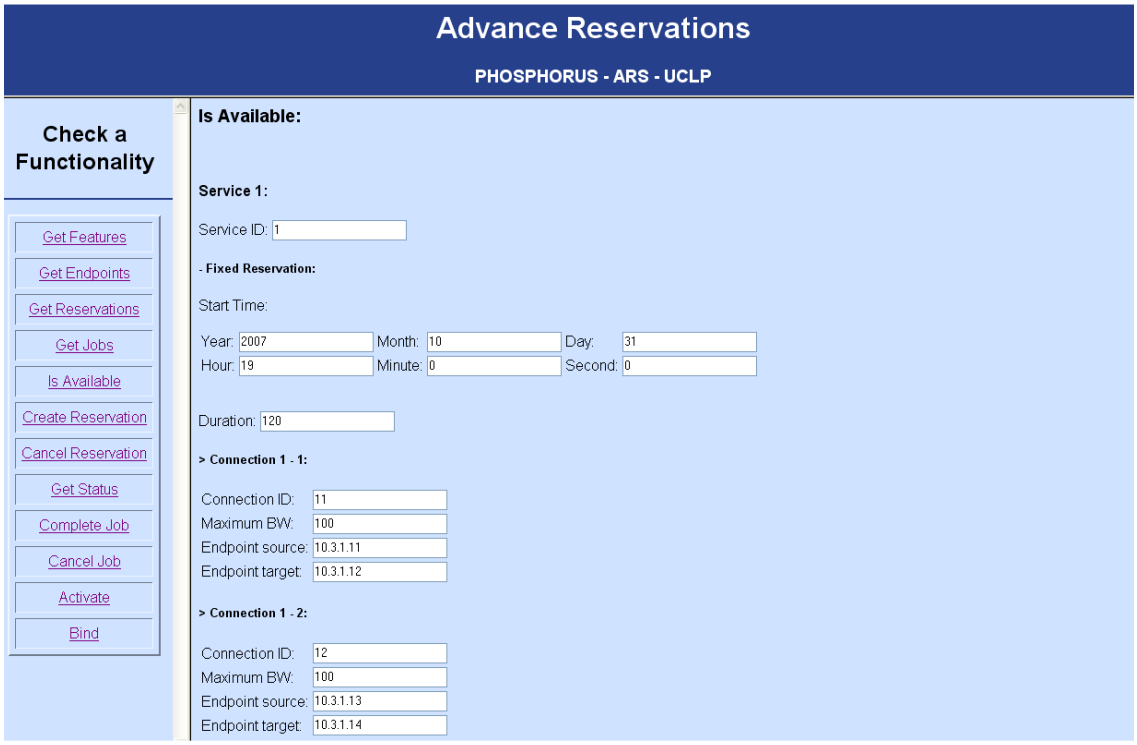


Figure 81. Endpoints of the network

IV. Is Available

To request the availability of a reservation the user must fill in the gaps the information required to create the reservation: the identifier of the service, the start time and duration, the identifier of the connection/s, the bandwidth and the IP address of the target and source endpoints. In this example, a reservation with three services is created. The first service is requested on 31th October 2007 at 19:00 and has two connections, one from Lleida to Tarragona and the other from Barcelona to Girona, and the other ones have one connection each one; the second service is requested from Lleida to Barcelona at the same time than the first service, and the third service is requested at 19:30 from Tarragona to Barcelona.



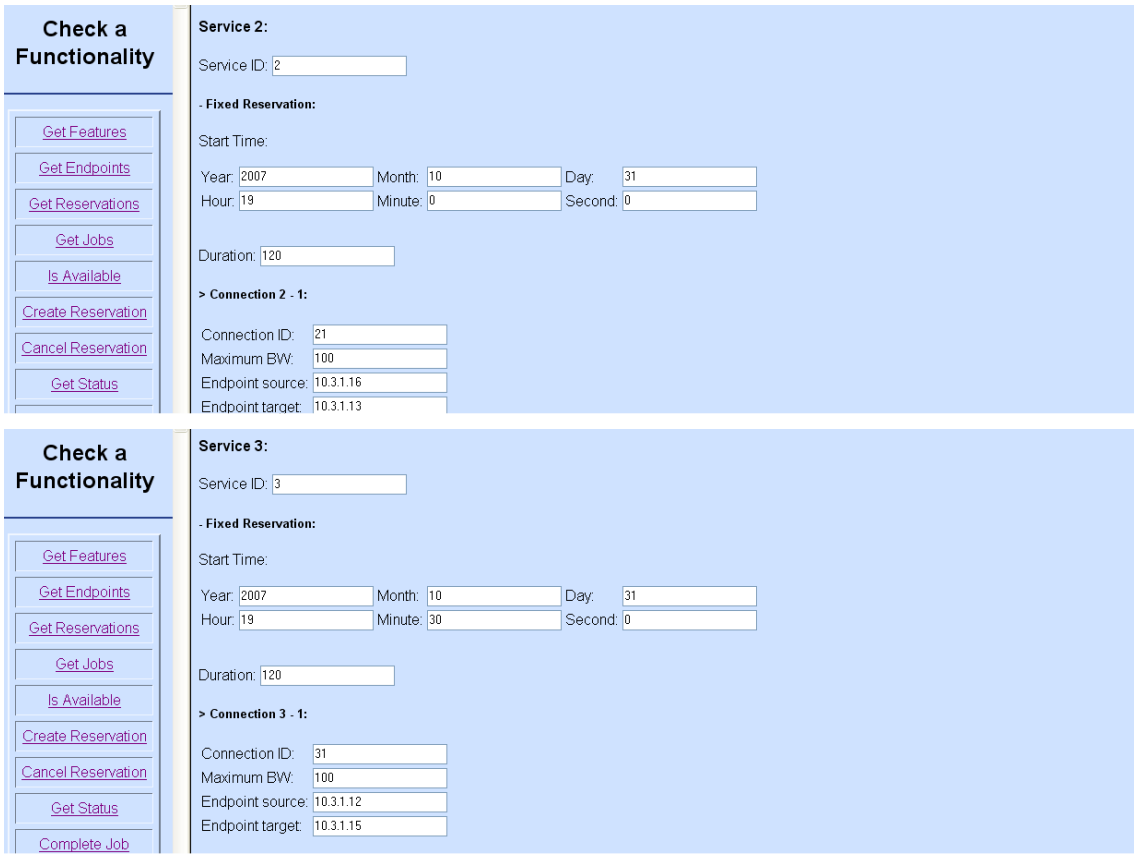


Figure 82. Is Available request

Once the request has been done, the response shows the start time requested for each service and the availability of each connection requested. In this example, the first two services are incompatible because request the same endpoint at the same time for two different connections. So, for the first two services the availability is 1 for the first service (source port unavailable because 10.3.1.13 is incompatible with the second service) and 2 for the second service (destination port unavailable because the same endpoint 10.3.1.13 is requested); for the third service the availability is 0 because this connection is possible. As the reservation request is not available because not all the connections are available, an alternative start time can't exist because of the incompatibility, no alternative start time is returned.

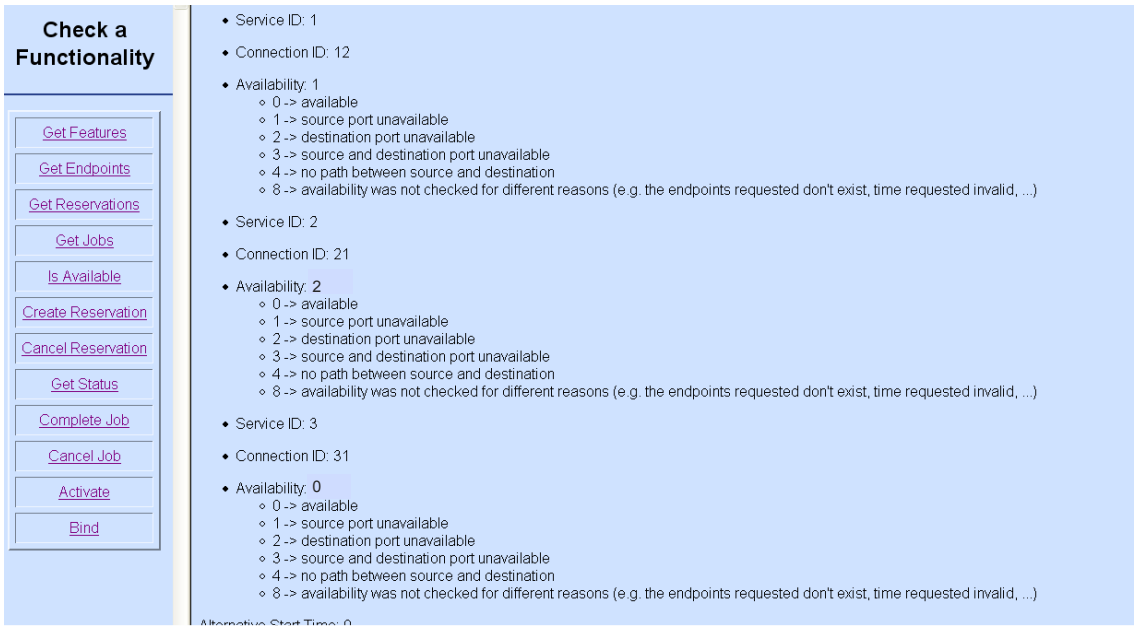
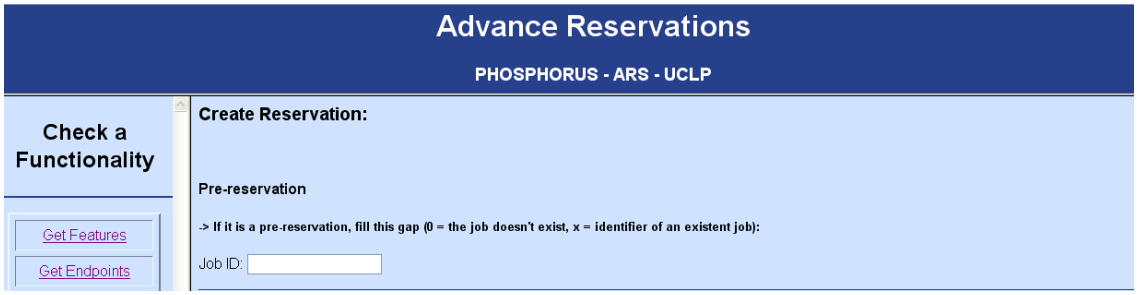
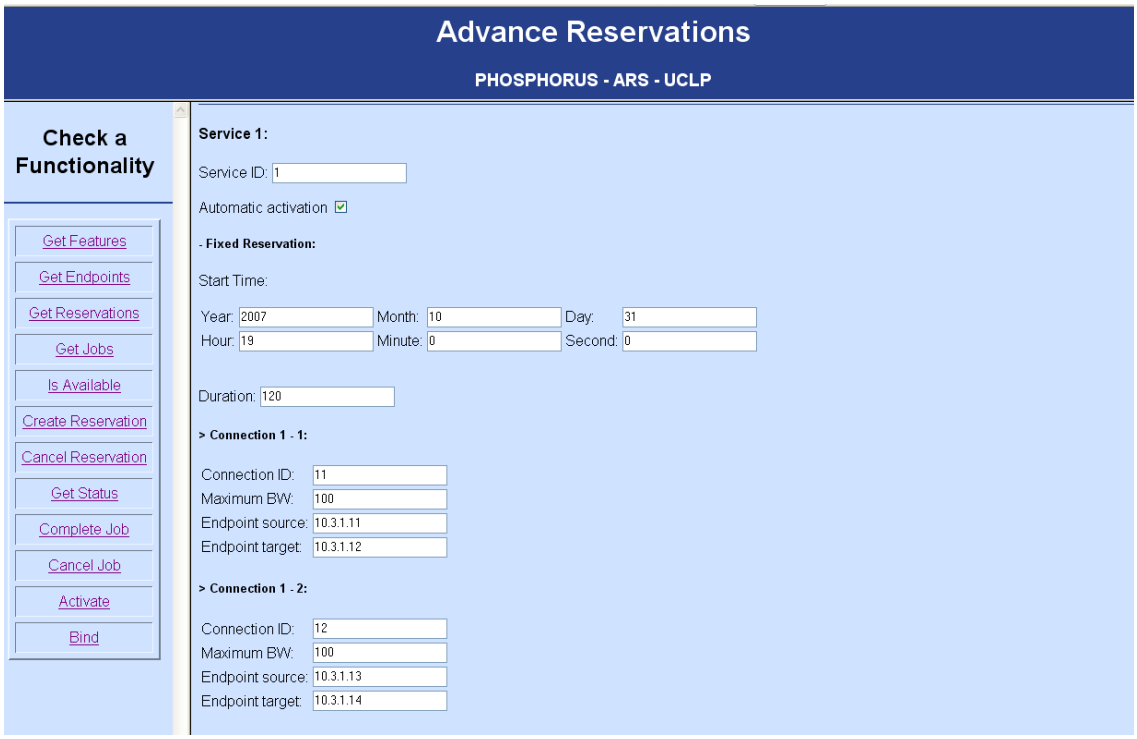


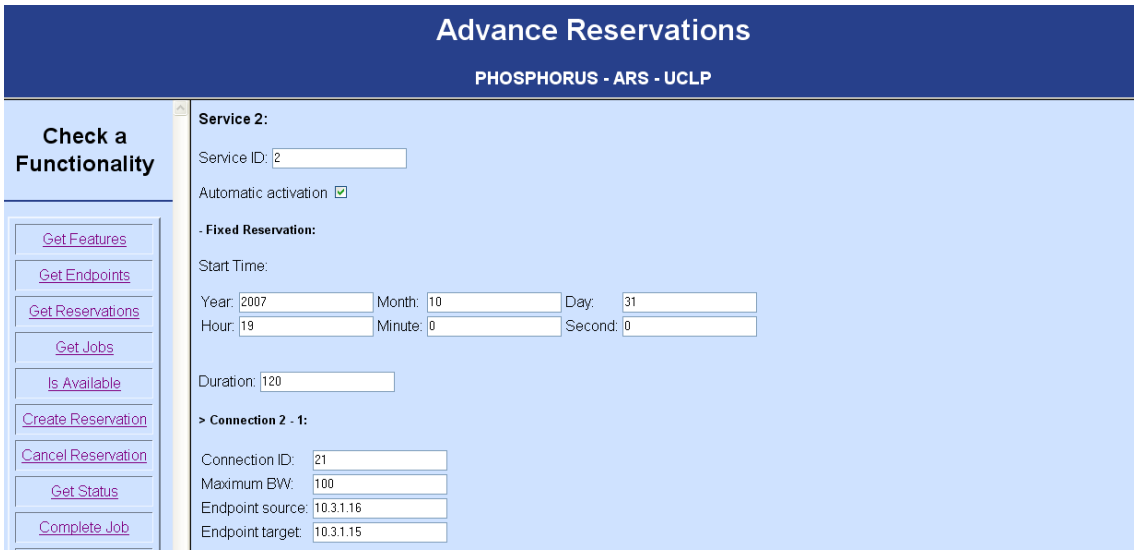
Figure 83. Is Available Response

V. Create Reservation

In the following screenshots the creation of a permanent reservation is requested. As the reservation is permanent, no jobID is indicated. The reservation has three services, the first one has two connections and the other ones have one connection each one. The first service is requested on 31th October 2007 at 19:00 and has two connections, one from Lleida to Tarragona and the other from Barcelona to Girona, and the other ones have one connection each one; the second service is requested from Lleida to Barcelona at the same time than the first one, and the third service is requested at 19:30 from Tarragona to Barcelona.




Advance Reservations
PHOSPHORUS - ARS - UCLP
Check a Functionality
Get Features
Get Endpoints
Get Reservations
Get Jobs
Is Available
Create Reservation
Cancel Reservation
Get Status
Complete Job
Cancel Job
Activate
Bind
Service 1:
Service ID: 1
Automatic activation
- Fixed Reservation:
Start Time:
Year: 2007 Month: 10 Day: 31
Hour: 19 Minute: 0 Second: 0
Duration: 120
> Connection 1 - 1:
Connection ID: 11
Maximum BW: 100
Endpoint source: 10.3.1.11
Endpoint target: 10.3.1.12
> Connection 1 - 2:
Connection ID: 12
Maximum BW: 100
Endpoint source: 10.3.1.13
Endpoint target: 10.3.1.14


Advance Reservations
PHOSPHORUS - ARS - UCLP
Check a Functionality
Get Features
Get Endpoints
Get Reservations
Get Jobs
Is Available
Create Reservation
Cancel Reservation
Get Status
Complete Job
Service 2:
Service ID: 2
Automatic activation
- Fixed Reservation:
Start Time:
Year: 2007 Month: 10 Day: 31
Hour: 19 Minute: 0 Second: 0
Duration: 120
> Connection 2 - 1:
Connection ID: 21
Maximum BW: 100
Endpoint source: 10.3.1.16
Endpoint target: 10.3.1.15

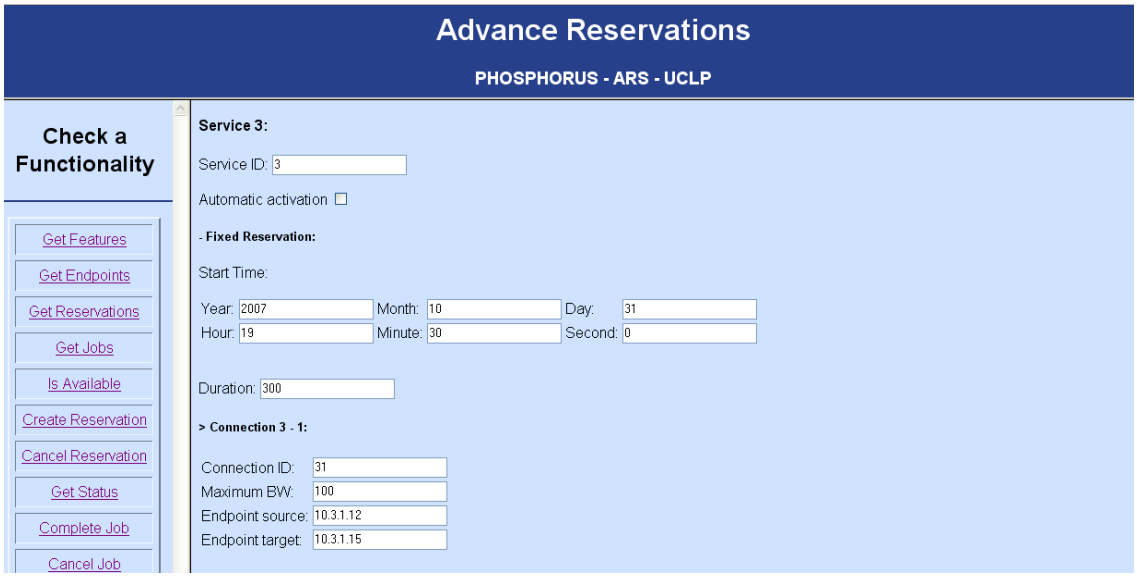


Figure 84. Create Reservation request

The response shows the time requested for each service and if the reservation is possible, returns the identifier of the reservation created. In this case the reservation is possible and its identifier is 3.

Figure 85. Create Reservation response

VI. Get Reservations

To get the reservations stored in the database, the user must click on the link ‘Get Reservation’ and click on the button ‘Accept’.

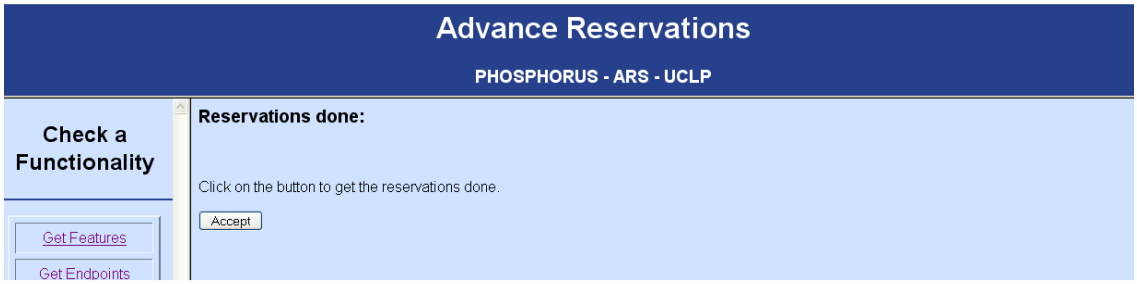


Figure 86. Get Reservation request

The system shows the reservations done. In this case the reservation requested in the last example is shown. The information returned of each reservation is the identifiers of the reservation, services and connections, a Boolean that indicates if the reservation is permanent or pre, the status of each service and connection, if the service must be activated automatically, the start time and duration of each service and the endpoints path of each connection indicating for each endpoint its identifier and the interface type (note that the internal endpoints hasn't got interface type).

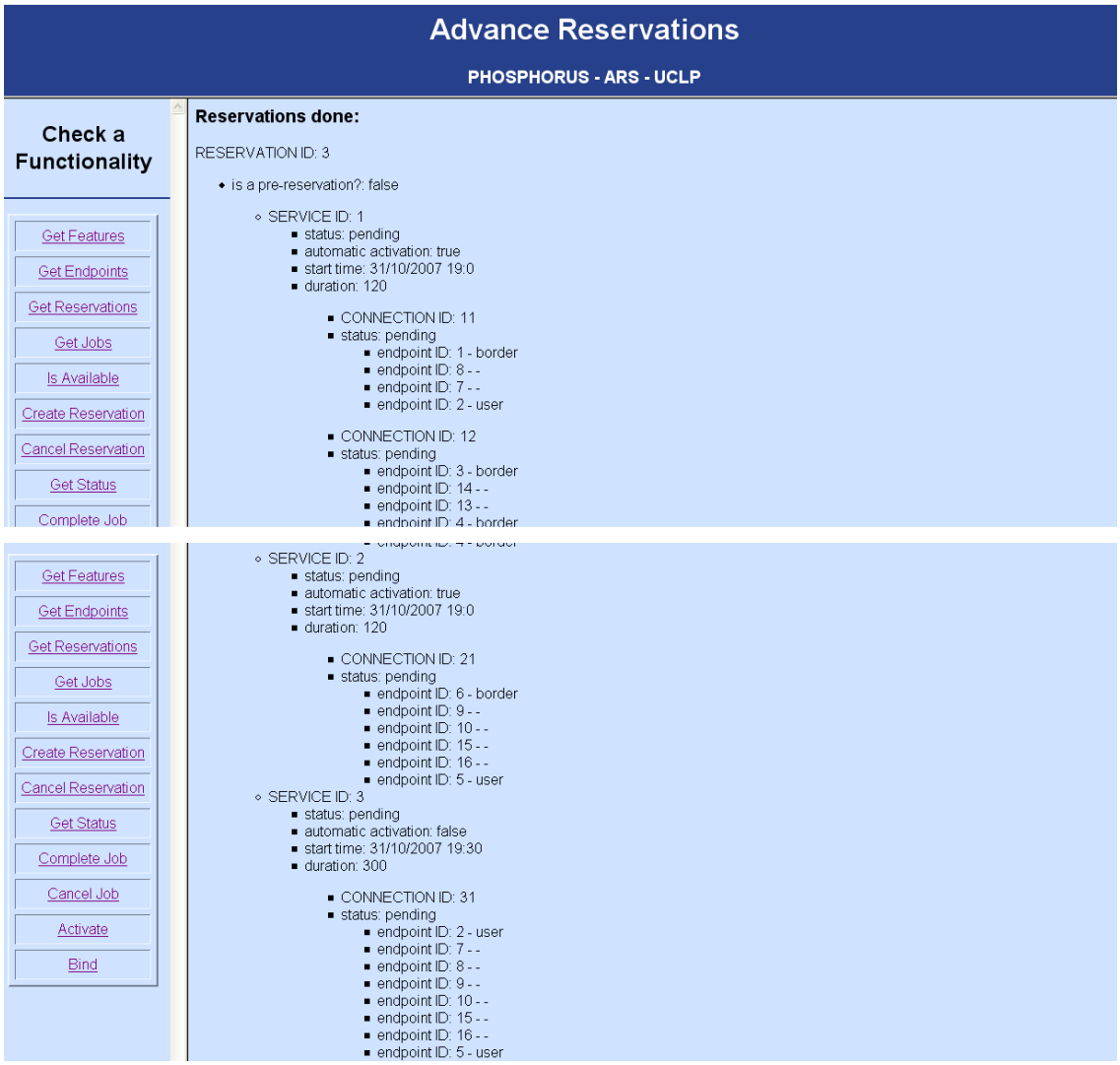


Figure 87. Reservations done

VII. Get Status

To get the status of the services of a reservation, the user must click on the link 'Get Status', fill in the gaps and click on the 'Accept' button. In this case the statuses of the three services of the reservation are requested. The user must introduce the identifier of the reservation and the services in the gaps.

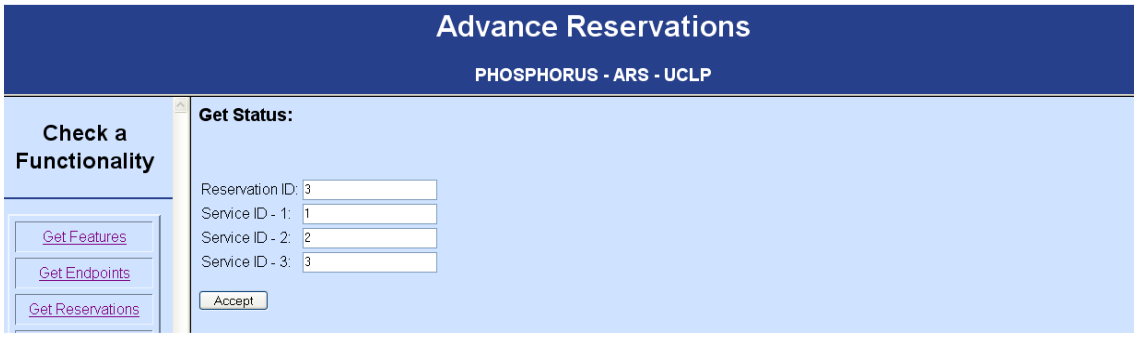


Figure 88. Get Status request

Once the request is processed, the system shows the status of the connections of the services requested. For each service the identifier of the service, the status, the automatic activation flag, the start time and duration, the connection identifier, the status of the connection and the source and target endpoints of the connection are shown. In this example, as the start time requested has not arrived, the status of all the services is 'pending'.

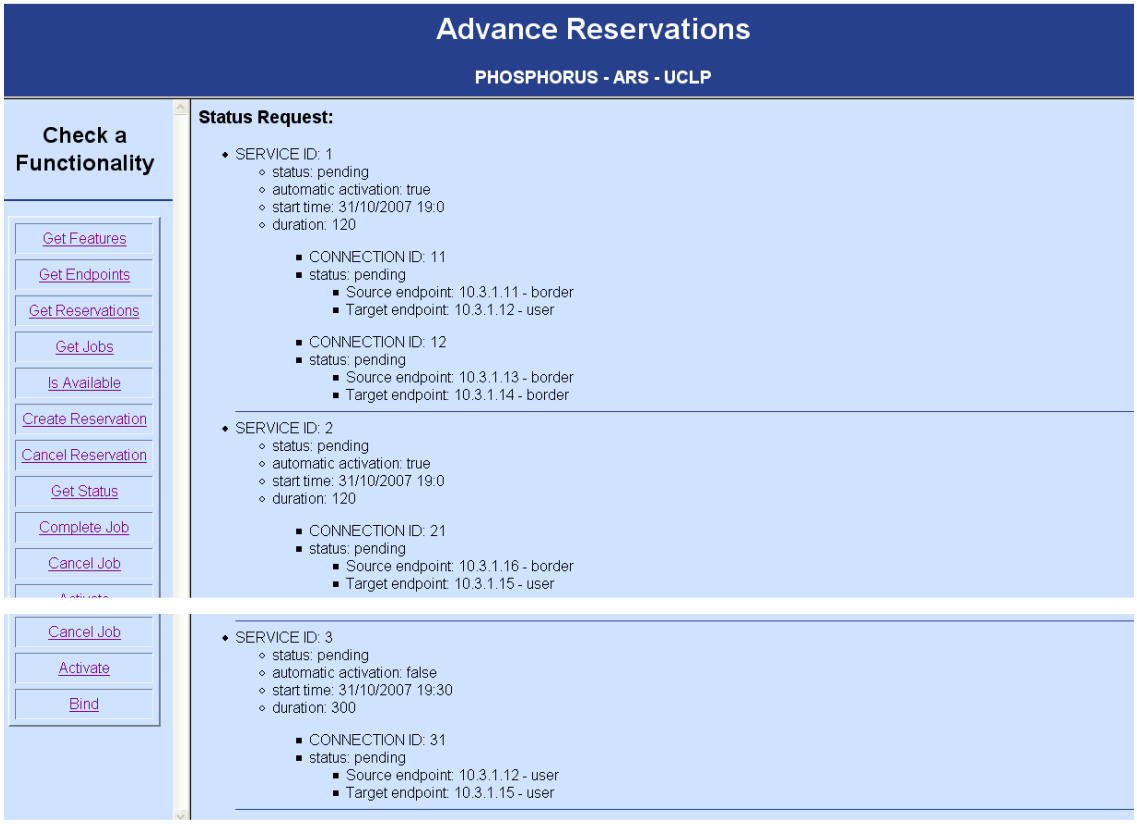


Figure 89. Get Status response

VIII. Activate

To activate a service of a reservation, the user must click on the link 'Activate', fill in the gaps and click on the 'Accept' button.

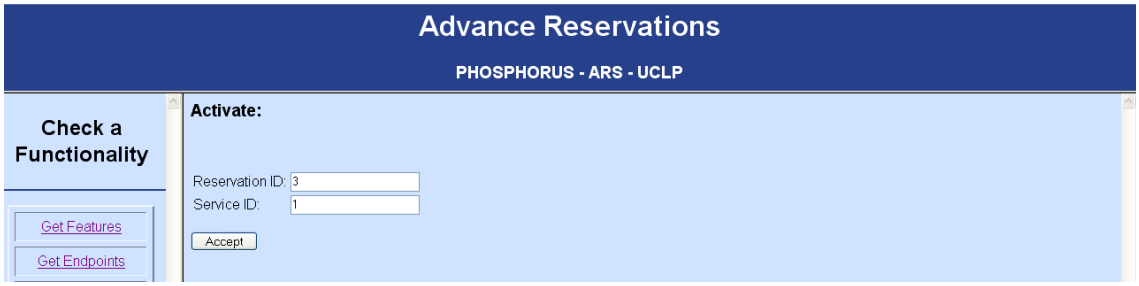


Figure 90. Activate

If the start time requested for the service has not arrived, the scheduler is programmed; if the start time has passed, as soon as possible, the connections of the service will be activated physically and the statuses will be set as 'active'. When its end time will arrive (start time calculated plus the duration requested), these connections will be deleted and the statuses will be set as 'completed'.

Figure 91. Activate Response

IX. Cancel Reservation

When the user wants to cancel a reservation, he must click on the link 'Cancel Reservation', indicate the identifier of the reservation he wants to cancel and finally, click on the 'Accept' button.

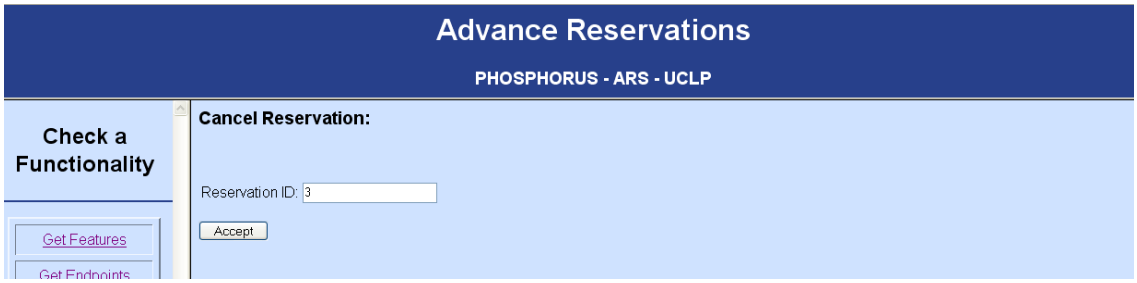


Figure 92. Cancel Reservation request

If the reservation exists and the cancellation has been successfully a message indicating it is returned.

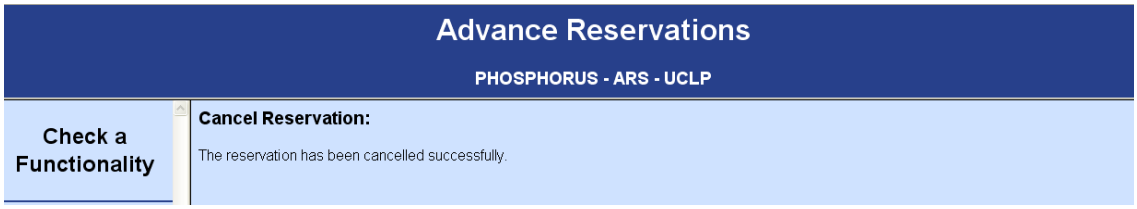


Figure 93. Cancel Reservation response

X. Create Pre-reservation

In the following example a pre-reservation will be created. In this case the job doesn't exist, so it must be set as zero the identifier of the job. The pre-reservation created contains one service requested at 17:00 and one connection from Barcelona to Tarragona. The service won't be activated automatically.

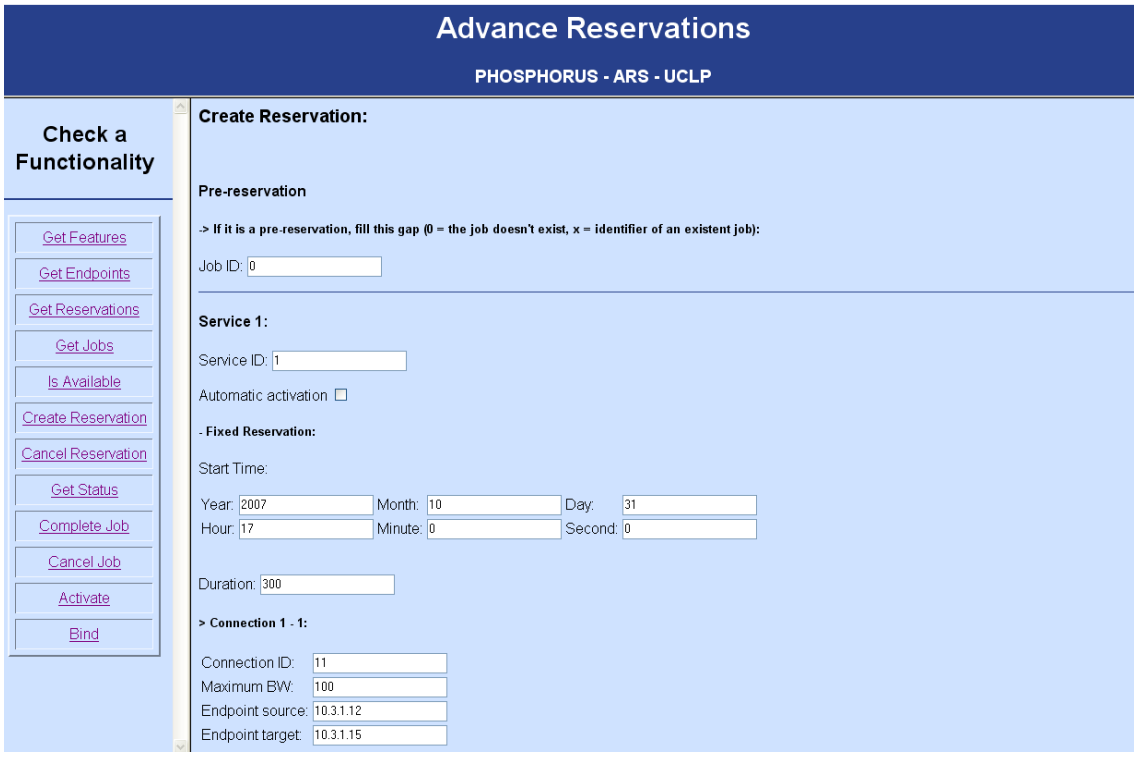


Figure 94. Create Pre-Reservation request

Once the request is processed, if the creation is possible the system returns the identifier of the reservation and the job created. In this case the identifier of the reservation is 4 and the one of the job is 1.

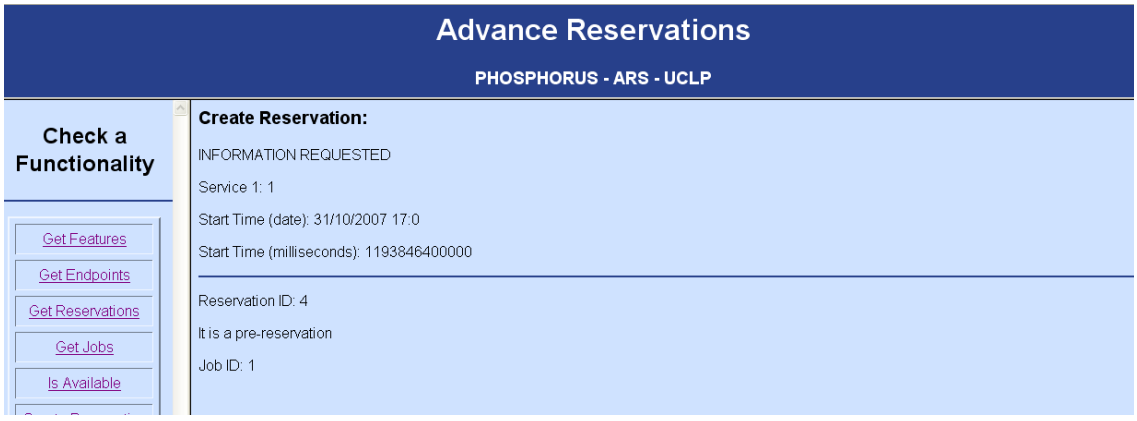


Figure 95. Create Pre-Reservation response

XI. Get Jobs

To see the jobs existing in the database, the user must click on the link 'Get Jobs' and after click on the 'Accept' button.

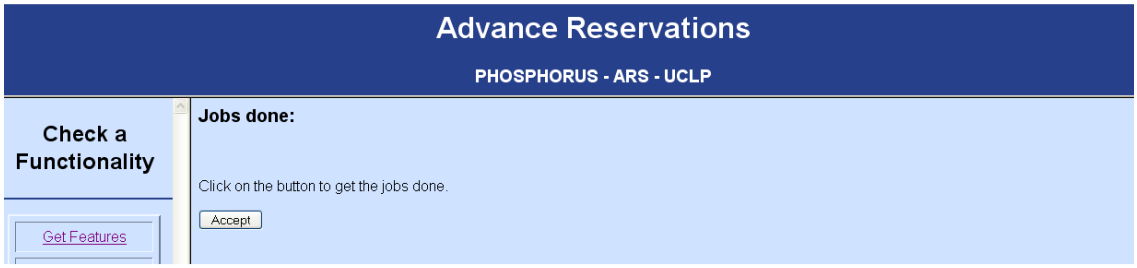


Figure 96. Get Jobs request

In this example, the pre-reservation created previously is shown. The same information as the operation 'Get Reservations' is shown, but in this case the identifier of the job is also shown.

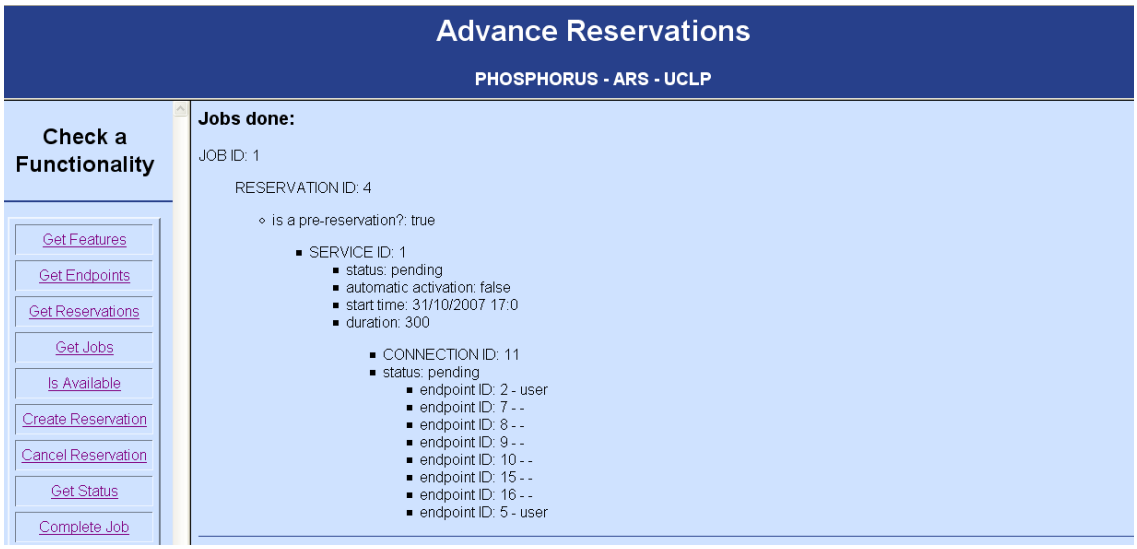


Figure 97. Get Jobs response

XII. Complete Job

To complete a job, the user must click on the link 'Complete Job', indicate the identifier of the job and click on the 'Accept' button.

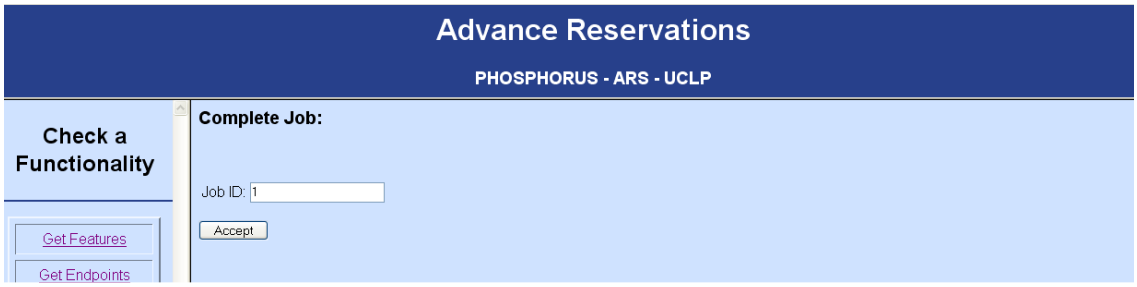


Figure 98. Complete Job request

If the job has been completed successfully, a message indicating it is returned.

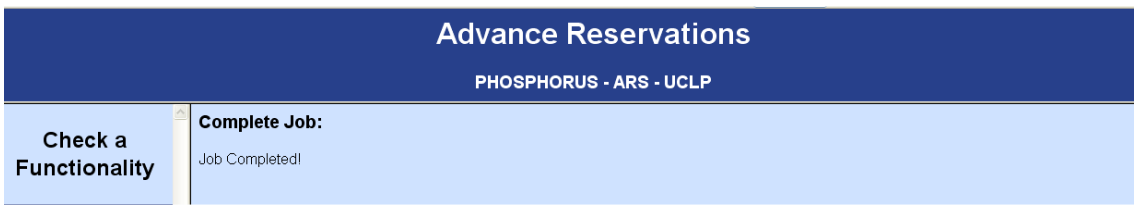


Figure 99. Complete Job Response

Making a 'Get Reservations' requested it can be seen that the pre-reservation created previously is now a permanent reservation.

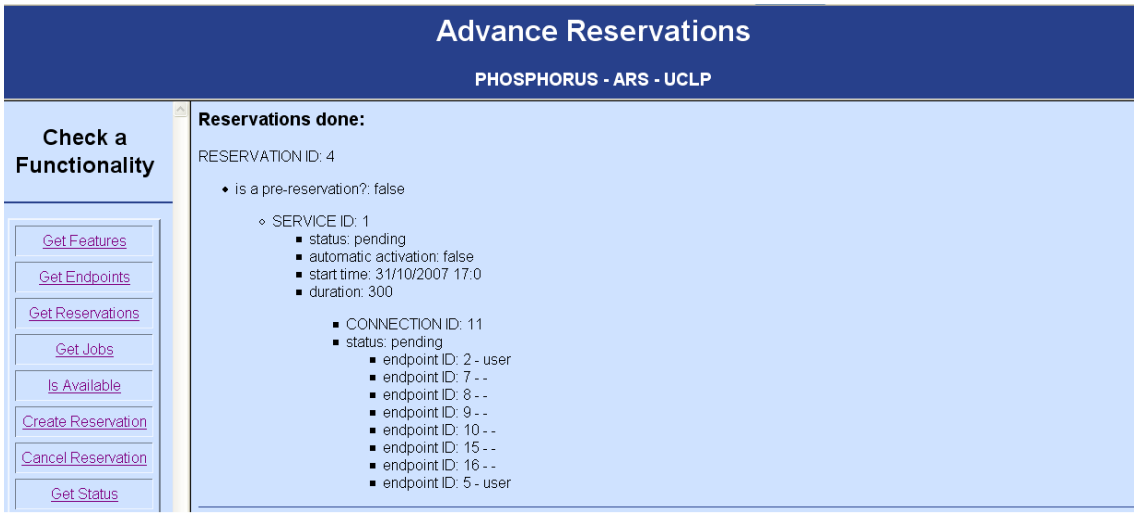


Figure 100. Reservations done (after complete Job)

XIII. Cancel Job

To cancel a job the user must click on the link 'Cancel Job', indicate the identifier of the job to be cancelled and click on the 'Accept' button.

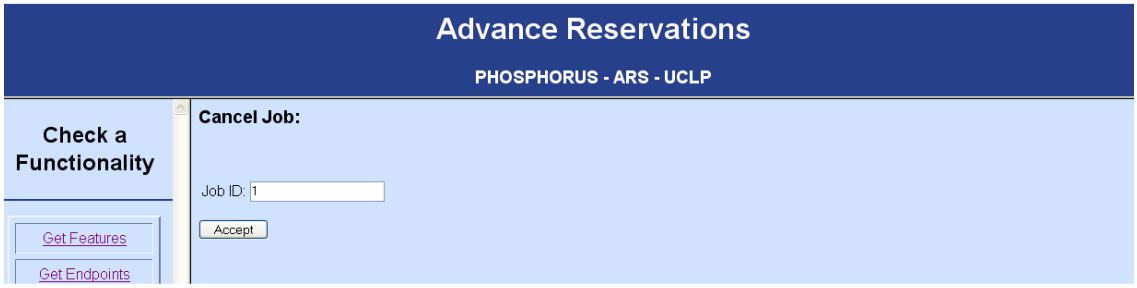


Figure 101. Cancel Job Request

If the cancellation has been successful a message indicating it is returned.

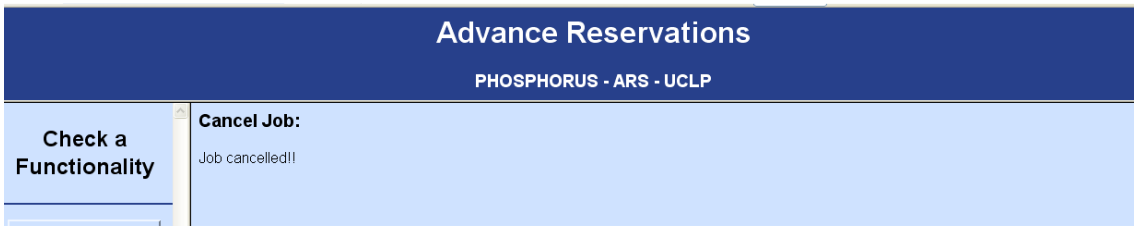


Figure 102. Cancel Job response successful

If the job does not exist, a message indicating it is returned. In this case as the job was already completed, there were no jobs to cancel.

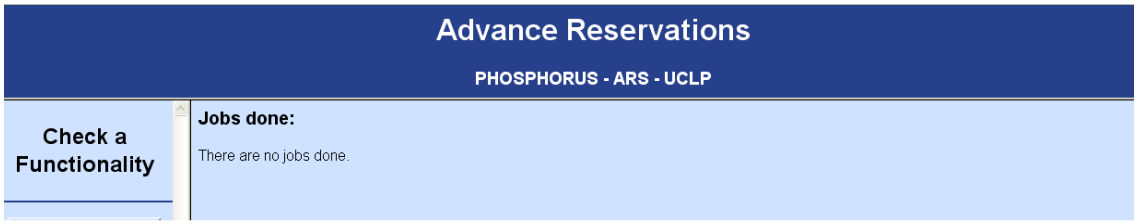


Figure 103. Cancel Job response no successful

11. DEMONSTRATIONS

11.1. SC07

The SC is the annual congress about supercomputing. This year has been done in Reno, Nevada, USA. SC07 is the international conference for high performance computing, networking, storage and analysis.

In this conference, a demonstration of the PHOSPHORUS WP1 has been performed on 13th November and it has been successful.

In the demonstration of the PHOSPHORUS IST FP6, two connections were established:

- One connection from the Client1 to a KoDaVis server (a streaming video service of the VIOLA testbed).
- Another connection from the Client2 to the KoDaVis server.

These two connections were created by means of an immediate reservation of the network through the CRC, i2CAT and VIOLA domains.

In the following picture the test scenario of the demonstration can be seen. On the left side there are the two Clients, located at Poland, in the middle the CRC (Canada) and i2CAT(Spain) domains and on the right side the VIOLA (Germany) domain with its service KoDaVis.

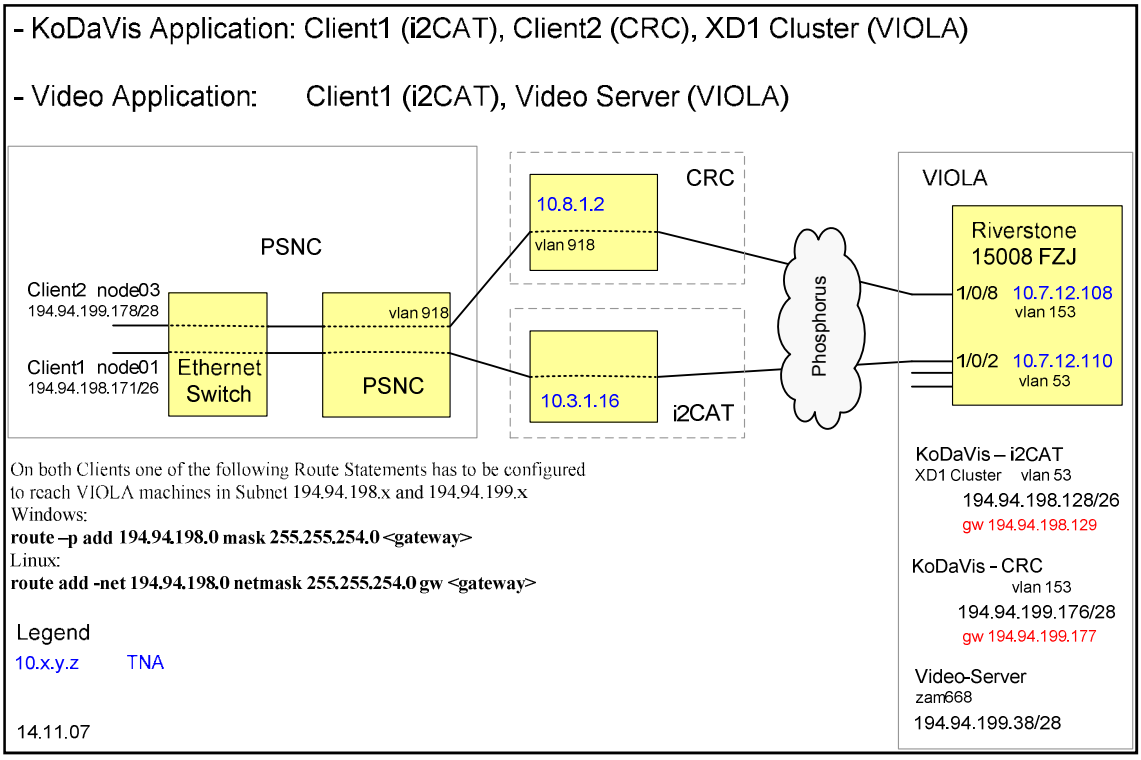


Figure 104. Test Scenario

In the following picture the software modules used to the perform de demonstration can be seen.

A web application is used to perform the demonstration. This figure is divided as the different layers of the PHOSPHORUS: Service plane, Control plane, Data plane and the network. The different systems integrated in the PHOSPHORUS project can be

seen in the figure (ARGON, UCLP and DRAC) and also its adapters connected to the Manager of the PHOSPHORUS project that allows the interconnection of these different systems. The discontinuous arrows show where the ARS is located (between the UCLP and the adapters of the i2CAT and the CRC).

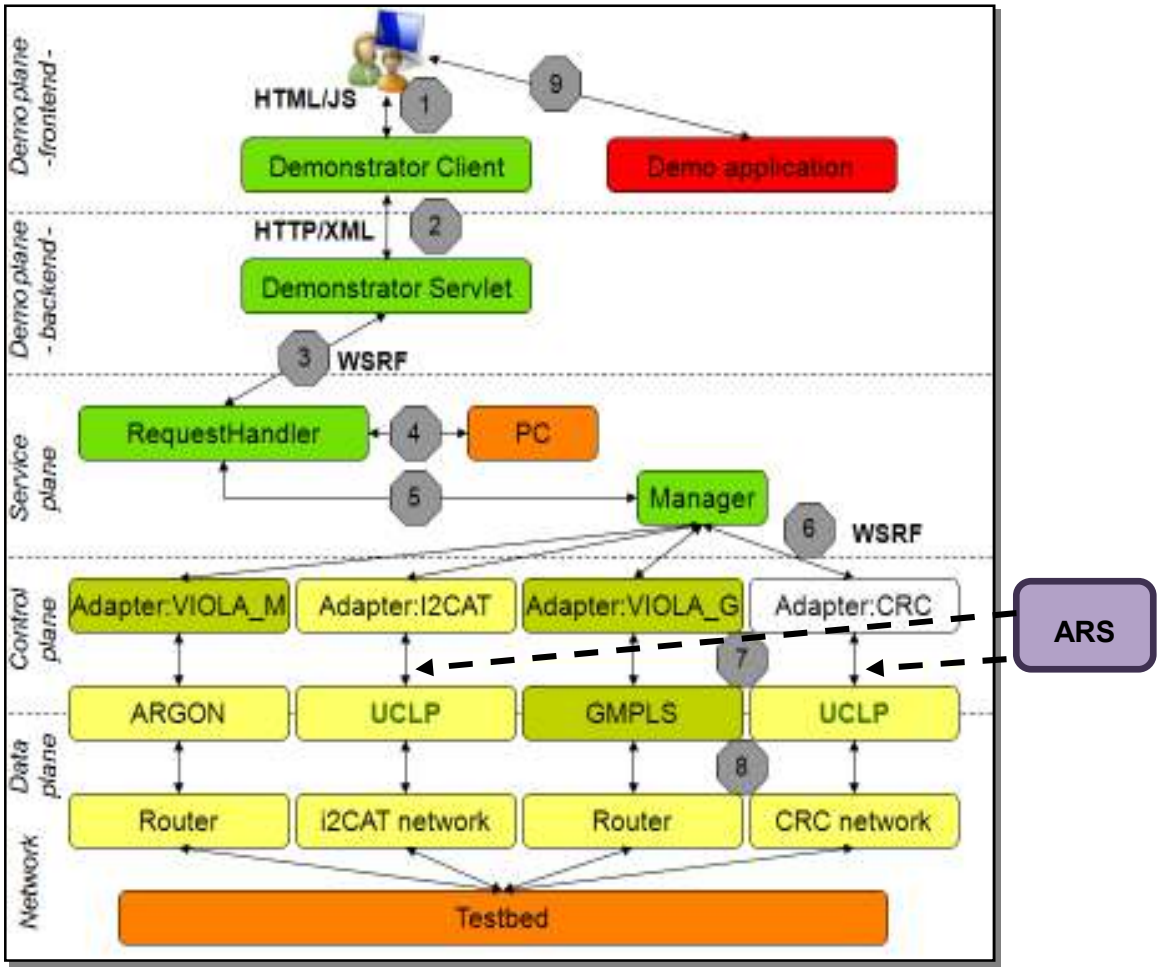


Figure 105. Software Modules

In the following picture, an image of the i2CAT network is shown. The part controlled by the ARS is the one in the green square (Under UCLP control).

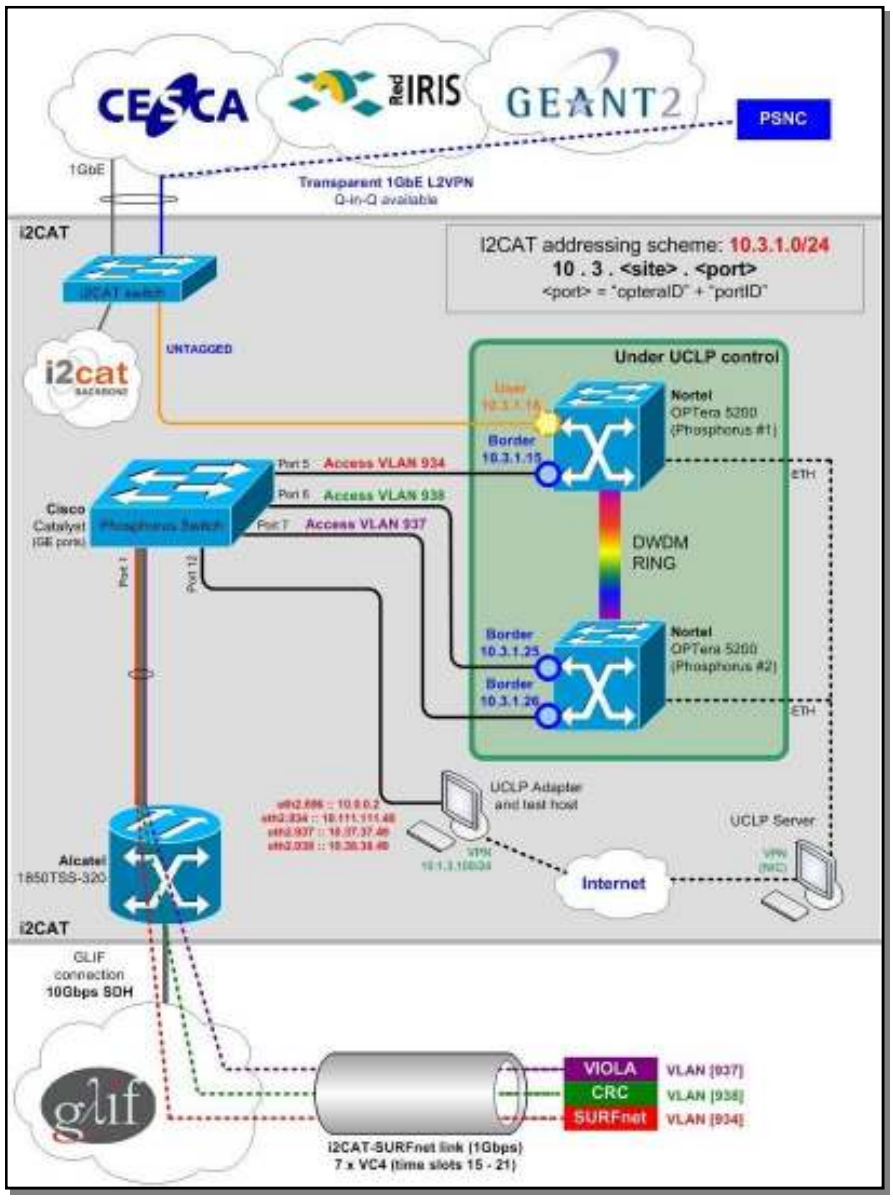


Figure 106. i2CAT network

In the following picture, an image of the CRC network is shown. The entire network is controlled by the ARS.

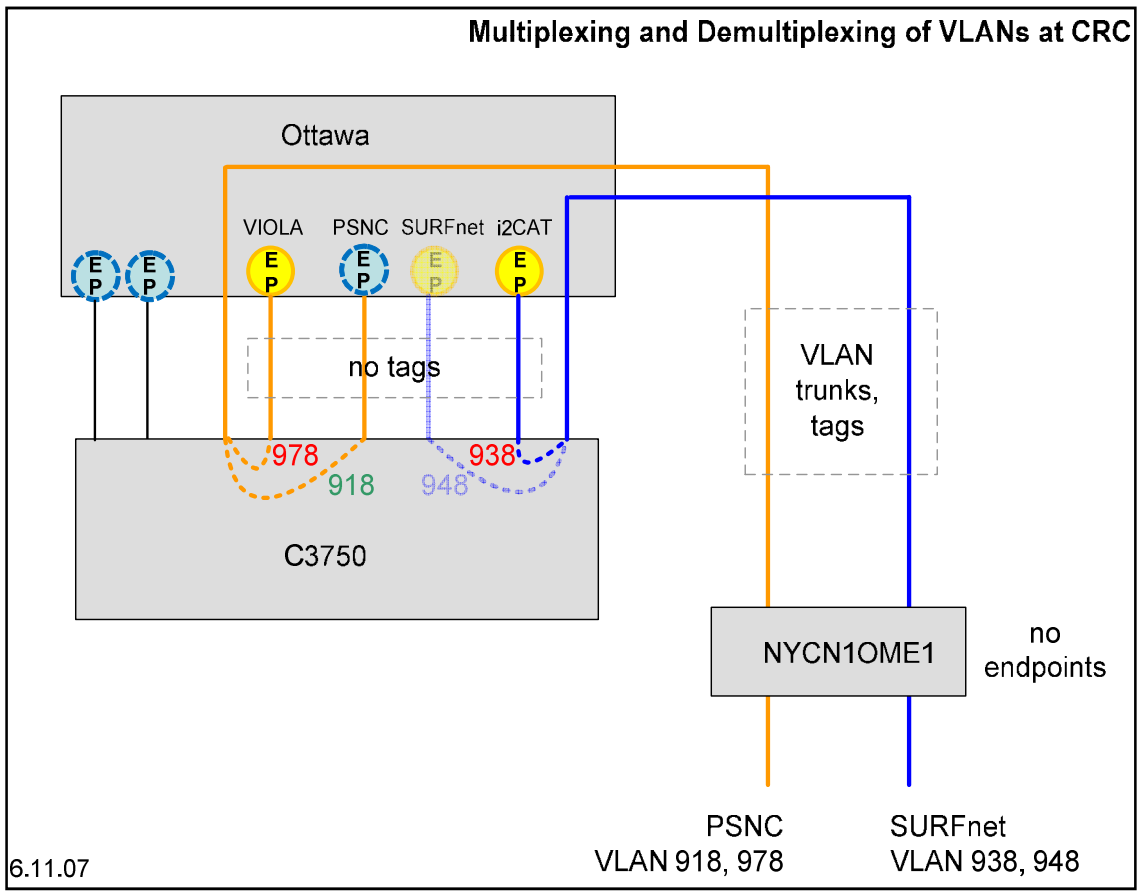


Figure 107. CRC network

11.1.1. Workflow

To perform the demonstration, the workflow used has been the following one: Firstly, an empty Google Map was shown. Next, this map was updated with all the endpoints and testbeds and preconfigured domain informations were added via the demonstrator client. After, links were added in the network and the updated Google Map was shown again, also preconfigured link informations were added via the demonstrator client. Once the map was configured, a simple immediate reservation through all domains was created and the updated Google Map was shown (in the figure below the reservation done can be seen). Next, the demostrator application was started (a video was sent through the network, from the KoDaVis to the Client 1 and 2) and the source and destination points on the Google Map were shown. Finally, the reservation was cancelled (the application stopped working) and the updated Google Map was shown again.

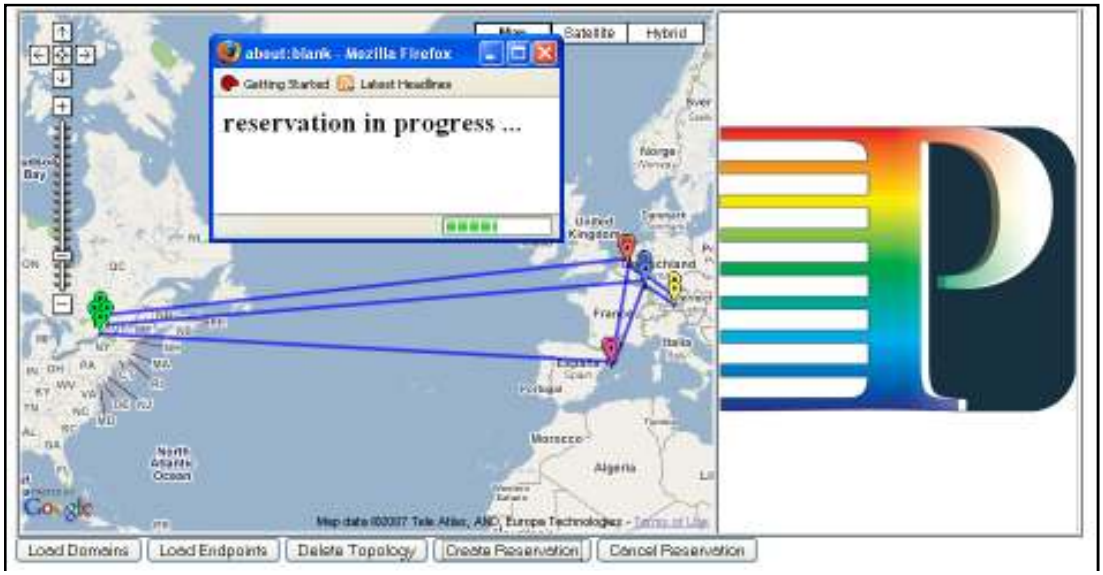


Figure 108. Demonstrator web application

11.2. EC REVIEW

The next event where a demonstration of the PHOSPHORUS has been performed is the EC Review on 13th December, 2007.

12. FUTURE WORK

Planned future work can be split in two main topics. The first topic is about introducing the deferrable and malleable reservations in the ARS system; in this way more possibilities of making reservations will be offered to the users. The second topic is about offering more services to the users, such us mark a reservation as 'cancelled by the user' instead of deleting it from the database, delete from the database a group of reservations or all the reservations, etc; and the same for the jobs.

At the moment, as it is indicated in the specification of the project, the user must be authenticated every time he wants to request some functionality. In the future, this authentication will be global for the user; that is, the user will be authenticated just once every time he starts the service.

Finally, the web application presented in the last section (Testing) may be improved in order to future users (mentioned in section 4.2) will be able to use it to interact directly with the ARS.

13. CONCLUSIONS

13.1. CONCLUSIONS OF THE PROJECT

In this document it has been described a service that gives the functionality to the UCLP system of supporting advance reservations using an interface that allows the communication between PHOSPHORUS and UCLP.

Before the creation of the ARS, UCLP could not offer its services to users that required an advance reservation service, but from now on, UCLP will have more possible different users because its functionalities have been extended and improved. Moreover, thanks to these developments, the UCLP system can be integrated in the PHOSPHORUS project.

To conclude, it must be said that this prototype has been implemented and tested within the i2CAT and CANARIE networks and its working is properly. Moreover, the ARS system has been proved in the international demonstration of the PHOSPHORUS in the SC07 on 13th November with success and it will be proved again in an European demonstration of the PHOSPHORUS in the EC Review on 13th December. Thus, these demonstrations validate this project.

I would like to add that a paper based on the ARS system has been submitted to the ONDM 2008, the 12th International Conference on Optical Networking Design and Modelling that will be held in Vilanova i la Geltrú, Catalonia, Spain, on March 12-14, 2008.

13.2. PERSONAL CONCLUSIONS

The elaboration of this project has given me the opportunity to think about different aspects of the university degree and evaluate what I have learnt, what I must learn in the future and what I have had to learn to develop this project, because there weren't specific courses in the degree about the topics of the project, or because I haven't done these specific courses. At times I thought that I wouldn't be able to achieve it because my experience in this field was really limited; however, I am satisfied with the work I have done.

I would like to add that doing a final project of the university degree is a hard experience in many times, but very satisfactory in general and so much didactic.

Developing a system related to two international projects (UCLP and PHOSPHORUS IST FP6) is a very interesting experience, because you learn from problems, delays, incompatibilities… normal facts that happen in these contexts. Working in this environment with people from all over the world and that have studied different things, has helped me to understand better how the project in the real world is and I think that this will help me in the future, not only from a personal view, but also from a professional one.

Finally, the realization of this project has helped me to evaluate the degree in a constructive way; in other words, I have learned that in the degree everything is not taught, the degree teaches you how to learn for itself, to think and to wise up.

GLOSSARY

Agile Software Development	It is a conceptual framework for software engineering that promotes development iterations throughout the life-cycle of the project. There are many agile development methods; most minimize risk by developing software in short amounts of time. Software developed during one unit of time is referred to as an iteration, which may last from one to four weeks. Each iteration is an entire software project: including planning, requirements analysis, design, coding, testing, and documentation. An iteration may not add enough functionality to warrant releasing the product to market but the goal is to have an available release (without bugs) at the end of each iteration. At the end of each iteration, the team re-evaluates project priorities. Agile methods emphasize face-to-face communication over written documents.
Capex	Capital expenditures are expenditures creating future benefits. A capital expenditure is incurred when a business spends money either to buy fixed assets or to add to the value of an existing fixed asset with a useful life that extends beyond the taxable year.
GÉANT	It is the main European multi-gigabit computer network for research and education purposes. GÉANT link speeds range from 155 Mbit/s on the slowest spur links to 10 Gbit/s in the core optical fiber network.
GÉANT2	It is the seventh generation of pan-European research and education network, successor to the pan-European multi-gigabit research network GÉANT
GMPLS	It enhances MPLS architecture by the complete separation of the control and data planes of various networking layers. GMPLS enables a seamless interconnection and convergence of new and legacy networks by allowing end-to-end provisioning, control and traffic engineering even when the start and the end nodes belong to heterogeneous networks. GMPLS is based on the IP routing and addressing models.
Grid Computing	"Distributed" or "grid" computing in general is a special type of parallel computing which relies on complete computers (with onboard CPU, storage, power supply, network interface, etc.) connected to a network (private, public or the Internet) by a conventional network interface, such as Ethernet.
JDBC	Java DataBase Connector. It is an API for the Java programming language that defines how a client may access a database. It provides methods for querying and updating data in a database. JDBC is oriented towards relational databases.
JSP	Java Server Page. It is a Java technology that allows software developers to dynamically generate HTML, XML or other types of documents in response to a Web client request. The technology allows Java code and certain pre-defined actions to be embedded into static content.
JUnit	JUnit is a simple framework to write repeatable tests. It is an

	instance of the xUnit architecture for unit testing frameworks.
Lightpath	A lightpath is a direct network path from one computer to another for which a permanent or temporary connection of fibre-optic cables is configured without using routers. Traffic on a lightpath does not encounter any other traffic. It therefore reaches its destination without congestion.
Metascheduling	It is a computer software technique of optimizing computational workloads by combining an organization's multiple Distributed Resource Managers into a single aggregated view, allowing batch jobs to be directed to the best location for execution.
MySQL	Database Management System
Opex	Operational expenditure is an on-going cost for running a product, business, or system. Its counterpart, a capital expenditure (CAPEX), is the cost of developing or providing non-consumable parts for the product or system.
Packet switching	It is a communications paradigm in which packets (discrete blocks of data) are routed between nodes over data links shared with other traffic. In each network node, packets are queued or buffered, resulting in variable delay. This contrasts with the other principal paradigm, circuit switching, which sets up limited number of constant bit rate and constant delay connections between the nodes for their exclusive use for the duration of the communication.
PosgreSQL	Database Management System
Scheduler	An essential part of a computer operating system is the scheduler, which schedules processor time for each process to facilitate multitasking.
Servlet	It is a Java programming language class used to extend the capabilities of servers that host applications accessed via a request-response programming model. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by Web servers. For such applications, Java Servlet technology defines HTTP-specific servlet classes.
Spiral Model	It is a software development process combining elements of both design and prototyping-in-stages, in an effort to combine advantages of top-down and bottom-up concepts. Also known as the spiral lifecycle model, it is a systems development method (SDM) used in information technology (IT). This model of development combines the features of the prototyping model and the waterfall model. The spiral model is intended for large, expensive and complicated projects.
Testbed	It is a platform for experimentation for large development projects. Testbeds allow for rigorous, transparent and replicable testing of scientific theories, computational tools, and other new technologies.
Tomcat	application server developed at the Apache Software Foundation (ASF). Tomcat implements the Java Servlet and the JavaServer Pages (JSP) specifications from Sun Microsystems, providing an

	environment for Java code to run in cooperation with a web server. It adds tools for configuration and management but can also be configured by editing configuration files that are normally XML-formatted. Tomcat includes its own internal HTTP Server
Globus Toolkit 4	It is an open source toolkit for building computing grids developed and provided by the Globus Alliance.
UNICORE	It is a Grid computing technology that provides seamless, secure, and intuitive access to distributed Grid resources such as supercomputers or cluster systems and information stored in databases. The UNICORE technology is open source under BSD licence.
Unified Process	The Unified Software Development Process or Unified Process is a popular iterative and incremental software development process framework. The best-known and extensively documented refinement of the Unified Process is the Rational Unified Process or RUP. The Unified Process is not simply a process, but rather an extensible framework which should be customized for specific organizations or projects. The *Rational Unified Process* is, similarly, a customizable framework. As a result it is often impossible to say whether a refinement of the process was derived from UP or from RUP, and so the names tend to be used interchangeably.

REFERENCES

[1] Lars-Olof Burchard (2003, February). “Source Routing Algorithms for Networks with Advance Reservations”. Communication and Operating Systems, Technische Universitaet Berlin. ISSN 1436-9915.

[2] Jun Zheng, University of Ottawa. “Toward Automated Provisioning of Advance Reservation Service in Next-Generation Optical Internet”. Baoxian Zhang, Graduate University of the Chinese Academy of Sciences Hussein T. Mouftah, University of Ottawa.

[3] Chin Guok, David Robertson, Mary Thompson, Jason Lee, Brian Tierney and William Johnston. “Intra and Interdomain Circuit Provisioning Using the OSCARS Reservation System”. Energy Sciences Network. Berkeley, California 94720.

[4] Sumit Naiksatam, Silvia Figueira, Stephen A. Chiappari and Nirdosh Bhatnagar. “Modeling and Simulation of Advance Reservations in Dynamically Provisioned Optical Networks”. Department of Computer Engineering, Department of Applied Mathematics. Santa Clara University.

[5] Jun Zheng and H. T. Mouftah, Fellow, IEEE. “Supporting Advance Reservations in Wavelength-Routed WDM Networks”. Department of Electrical and Computer Engineering. Queen’s University. Kingston, ON K7L 3N6, Canada.

[6] Silvia Figueira, Neena Kaushik, Sumit Naiksatam,Stephen A. Chiappari, and Nirdosh Bhatnagar. “Advance Reservation of Lightpaths in Optical-Network Based Grids”. Department of Computer Engineering, Department of Applied Mathematics. Santa Clara University.

[7] The UCLP Project Website, http://www.uclp.ca

[8] The PHOSPHORUS Project Website, http://www.ist-phosphorus.eu

[9] L.-O. Burchard and R. Lüling. An Architecture for a Scalable Video-on-Demand Server Network with Quality-of-Service Guarantees. In Proceedings of the 5th Intl. Workshop on Distributed Multimedia Systems and Applications (IDMS), Lecture Notes in Computer Science, Springer, volume 1905, pages 132–143, 2000.

[10] PHOSPHORUS Deliverable reference number: D.1.1. Requirements and specifications of interfaces and architecture for interoperability between NRPS, GMPLS, Middleware. I2cat foundation.

[11] The ARGON project home page – “Allocation and Reservation in Grid-enabled Optinc Networks”

http://www.viola-testbed.de/fileadmin/VIOLA/reports/B2-4-1-allocation.pdf

[12] DRAC http://www.nortel.com/solutions/optical/collateral/nn110181.pdf

[13] PostgreSQL web page: http://www.postgresql.org/

[14] Hibernate web page: http://www.hibernate.org/

[15] Globus Alliance home page: http://www.globus.org/

[16] Open Sypmhony web page: http://www.opensymphony.com/quartz/

[17] Eduard Grasa, Sergi Figuerola, Albert Lopez. “ARTICULATED PRIVATE NETWORKS IN UCLP”. Technical University of Catalonia and i2CAT Foundation, Barcelona, Spain.

[18] Unicore Project home page: http://www.unicore.eu/

[19] Wikipedia: http://www.wikipedia.org

[20] Volere home page: http://www.volere.co.uk

[21] Java web page: http://java.sun.com

[22] Tomcat web page: http://jakarta.apache.com

[23] PostgreSQL web page: http://www.postgresql.org

[24] MySQL web page: http://www.mysql.com

[25] Dolors Costal, M. Ribera Sancho, Ernest Teniente. Enginyeria del Software: Especificació, 2nd Edition, Edicions UPC, 2003.

[26] Renaud Waldura web page: http://renaud.waldura.com/doc/java/dijkstra/

[27] Agile Software Development: http://agilemanifesto.org/