ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
ΤΜΗΜΑ ΔΙΔΑΚΤΙΚΗΣ ΤΗΣ ΤΕΧΝΟΛΟΓΙΑΣ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΥΠΗΡΕΣΙΟΣΤΡΕΦΕΙΣ ΑΡΧΙΤΕΚΤΟΝΙΚΕΣ ΠΛΗΡΟΦΟΡΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ ΚΑΙ ΥΠΗΡΕΣΙΕΣ ΙΣΤΟΥ ΣΤΟΝ ΤΡΑΠΕΖΙΚΟ ΤΟΜΕΑ

ΕΠΙΒΛΕΠΩΝ: Γ. ΒΑΣΙΛΑΚΟΠΟΥΛΟΣ

ΑΚΑΔ. ΕΤΟΣ 2007-08

ΜΥΤΙΛΗΝΑΙΟΥ ΕΛΕΝΗ

Contents

Part I

Background

Part I contains the following chapters:

- Chapter 1, Workflow Concepts
- Chapter 2, Resource Management Concepts
- Chapter 3, Authorization Policy Specification

1

Workflow Concepts

This chapter describes the key points of Workflow management systems (WFMS).

This chapter includes the following sections:

- Section 1.1, " Introduction to Workflow Concepts "
- Section 1.2, " The Case "
- Section 1.3, " The Task "
- Section 1.4, " The Process "
- Section 1.5, " Routing "
- Section 1.6, " Enactment "

1.1 INTRODUCTION TO WORKFLOW CONCEPTS

Workflow management systems (WFMS) are cooperative environments in which multiple distributed processing entities cooperate in order to accomplish tasks.

A workflow system automates a business process, enabling documents, information and tasks to be communicated among participants following a set of defined rules. It is regarded as an important means of increasing the productivity of enterprises since it crosses organisational boundaries and facilitates exchange of information among units and people. Hence, workflow technology is crucial to automating business processes and their reengineering. Workflow describes a business process in terms of a process model that links activities (either manual that necessitates user intervention or automatic) with participant users within the overall organisational framework. Now that data organization and user interfacing have largely disappeared from applications themselves, it seems that much of the software is devoted to business processes (procedures) and the handling of cases. Now, therefore, it has become attractive to isolate this component and find a separate solution for it. Not

only can this accelerate the development of information systems, it offers the added advantage that the business processes become easier to maintain.
The main workflow concepts are presented in the following subsections.

1.2 THE CASE

The primary objective of a workflow system is to deal with cases. Examples of cases include an insurance claim, a mortgage application, a tax return, an order or a patient in a hospital. Similar cases belong to the same case type. In principle, such cases are dealt with in the same way.

Each case has a unique identity. This makes it possible to refer to the case in question. A case has a limited lifetime. Between the appearance and disappearance of a case, it always has a particular state. This consists of three elements: (1) the values of the relevant case attributes; (2) the conditions which have been fulfilled; and (3) the content of the case.

A range of variables can be associated with each case. These case attributes are used to manage it. Thanks to them it is, for example, possible to indicate that a task may - under certain conditions - be omitted.

Conditions are used to determine which tasks have been carried out, and which still remain to be performed. Examples of conditions include 'order accepted', 'application refused' and 'under consideration'. We can also regard a condition as a requirement which must be met before a particular task may be carried out. Only once all the conditions for a task within a particular case have been met, can that task be performed. For any given case, it is clear at all times which conditions have been met and which not.

In general, the workflow system does not contain details of the content of the case, only those of its attributes and conditions. The content is contained in documents, files, archives and/or databases, which are not managed by the workflow management system.

1.3 THE TASK

The term task refers to one of the most important concepts in workflow modelling. By identifying tasks, it is possible to structure workflows. A task is a logical unit of work. It is indivisible and is thus always carried out in full.

Typing a letter, assessing a valuation report, filing a complaint, stamping a document and checking personal data are all examples of tasks. We can differentiate between manual, automatic and semi-automatic tasks. A manual task is entirely performed by one or more people, without any use of an application. By contrast, an automatic task is performed without any intervention by people. This usually means that an application - a computer program - can carry out the task entirely based upon previously-recorded data. Both a person and an application are involved in a semi-automatic task.

A task refers to a generic piece of work, and not to the performance of an activity for one specific case. In order to avoid confusion between the task itself and the performance of that task as part of a particular case, we use the terms work item and activity. A work item is the combination of a case and a task which is just about to be carried out. A work item is created as soon as the state of a case allows it. We can thus regard a work item as an actual piece of work which may be carried out. The term activity refers to the actual performance of a work item. As soon as work begins upon the work item, it becomes an activity. Note that, unlike a task, both a work item and an activity are linked with a specific case.

1.4 THE PROCESS

The way in which a particular category of cases should be carried out is contained in the relevant process. This indicates which tasks need to be carried out. It also shows the order in which this should be done. We can also regard a process as a procedure for a particular case type. In general, many different cases are carried out using a single process. It is therefore possible to enable a specific treatment, based upon the attributes of a certain case. For example, it may be that one task in the process is only performed on some of the cases. The order in which the tasks are performed may also vary, depending upon the properties of the case. Conditions are used to

decide which order is followed. In essence, a process is therefore constructed from tasks and conditions.

It is possible to make use of previously defined processes as part of another process. So, as well as tasks and conditions, a process may also consists of (zero or more) subprocesses. Each of the subprocesses again consists of tasks, conditions and possibly even further subprocesses. By explicitly identifying and separately describing subprocesses, frequently-occurring ones can be used repeatedly. In this way, complex processes can also be structured hierarchically. At the highest level of process description, we see a limited number of subprocesses. By examining one or more of these we can, as it were, 'zoom in' on particular sections of the process.

The lifecycle of a case is defined by a process. Because each case has a finite lifetime, with a clear beginning and end, it is important that the process also conforms to this. So each process also has a beginning and an end, which respectively mark the appearance and completion of a case.

1.5 ROUTING

The lifecycle of a case is laid down in the process. In this respect, we refer to the routing of the case. Routing along particular branches determines which tasks need to be performed (and in which order). In routing cases, we make use of four basic constructions:

- The simplest form of routing is the sequential performance of tasks. In other words, they are carried out one after the other. There is usually also a clear dependency between them. For example, the result of one task is necessary to the next.
- If two tasks can be performed simultaneously, or in any order, then we refer to parallel routing. In this case, there are two tasks which both need to be performed without the result of one affecting the other. The two tasks are initiated using an AND-split and later resynchronized using an AND-join.
- We refer to selective routing when there is a choice between two or more tasks. This choice may depend upon the specific properties of the case, as recorded in the relevant case attributes. Such a choice between alternatives is also

known as an OR-split. The alternative paths are reunited using an OR-join. As well as selective routing, we also use the terms alternative or conditional routing.

- In the ideal situation, a task is carried out no more than once per case. Sometimes, however, it is necessary to perform a particular task several times. Consider, for example, a task which needs to be repeated until the result of the subsequent 'check' task is satisfactory. We call this form of routing iteration.

1.6 ENACTMENT

A work item assignment can only be carried out once the state of the case in question allows it. But actual performance of such an assignment often requires more than this alone. If it has to be carried out by a person, they must first take the assignment from their 'in tray' before an activity can begin. In other words, only once the employee has taken the initiative, the work item is worked on. In such a case we refer to triggering: the work item is triggered by a resource (in the example, an employee). However, other forms of triggering are possible: an external event (for example, the arrival of a message) or reaching a particular time (for example, the generation of a list of orders at six o'clock). We thus distinguish between three types of trigger: (1) a resource initiative, (2) an external event, and (3) a time signal. Work items which must always be carried out immediately - without the intervention of external stimuli - do not require a trigger.

2

Resource Management Concepts

This chapter describes the key points of the management of resources and the link between a process definition and the resources available.

This chapter includes the following sections:

- Section 2.1, " Introduction to Resource Management Concepts "
- Section 2.2, " The Resource "
- Section 2.3, " Resource Classification "
- Section 2.4, " Allocating Activities To Resources "

2.1 RESOURCE MANAGEMENT CONCEPTS

Using the definition of a process, we can indicate which tasks need to be performed for a particular category of case. We can also show the order in which they must be carried out. However, the process definition does not indicate who should do it. But the way in which the work items are allocated to resources (people and/or machines) is very important to the efficiency and effectiveness of the workflow. In this section, we shall concentrate upon the management of resources and the link between a process definition and the resources available.

2.2 THE RESOURCE

A workflow system focuses upon supporting a business process. In this process, work is carried out by means of production, also called resources. In an administrative environment, the word resource primarily refers to office staff. However, a doctor, a printer, a doorman and an assembly robot are all examples of resources. The basic characteristic of a resource is that it is able to carry out particular tasks. We also assume that each resource is uniquely identifiable. And that it has a certain capacity..

2.3 RESOURCE CLASSIFICATION

In general, a resource can carry out a limited number of tasks. A task can usually only be performed by a limited number of resources. Because it is impracticable to indicate which resources are able to carry out each task, we classify them using resource classes. This is a group of resources. A resource may belong to more than one category. In general, we differentiate between two forms of resource classification: (1) that based upon functional properties and (2) that based upon position within the organization.

A functionally-based resource class is known as a role. It is also referred to as a function or qualification. A role is a group of resources, each of which has a number of specific skills. By linking a task to the correct role, one can ensure that the resource carrying it out is sufficiently qualified and authorized.

Resources can also be classified according to their place in the organization. A resource class based upon organizational rather than functional characteristics is also called an organizational unit. This form of classification can be used to ensure that a task is carried out at the right place in the organization. As already indicated, in most cases a resource classification consists of two parts. We call that part containing the functional structure the role model and that containing the organizational units the organization chart. Note that the term organization chart usually has a rather broader meaning, referring to the hierarchical structure of the organization.

2.4 ALLOCATING ACTIVITIES TO RESOURCES

In order to ensure that each activity is performed by a suitable resource, we provide each task in the process definition with an allocation principle. This specifies which preconditions the resource must meet. In most cases, the allocation specifies both a role and an organizational unit. The resource must then belong to the intersection between these two resource classes. However, it is also possible to define a much more complex allocation. The allocation may also depend upon the attributes of the case for which the task must be carried out. Depending upon these attributes we can, for example, select the organizational unit.

By making careful use of the case attributes, we can also ensure that an activity is performed by a specific resource. But the opposite is also possible.

By providing a task with an allocation principle, we specify the preconditions which the resource must meet. In most cases, there is more than one resource which may carry out the activity associated with a particular work item.

At the heart of a workflow system is the workflow engine. This ensures the actual enactment of a specified workflow. One of its core tasks is to allocate work items to resources. In doing so, it must take into account the resource classes specified, as well as such things as separation of duty and case management. In many cases, the workflow engine is nevertheless able to choose between a number of resources when allocating work. It then has to decide which will carry out the activity.

3

Authorization Policy Specification

This chapter provides a more detailed description of the management of resources and describes the authorization policy specification.

This chapter includes the following sections:

- Section 3.1, " Resource Management In More Detail "
- Section 3.2, " Allocation Principles - Authorization Policy Specification "
- Section 3.2.1, " Workflow Routing "
- Section 3.2.2, " Workflow Authorization "
- Section 3.2.3, " Context Sensitive Access Control "

3.1 RESOURCE MANAGEMENT IN MORE DETAIL

The allocation of resources to activities is not a simple issue. As we have seen, such concepts as the task, the case, the work item, the activity, the case attributes, the resource, the resource class, the role, the organizational unit and allocation are all closely connected with one another.

The preconditions formulated in the allocation policy can become highly complex. After all, an allocation relates tasks, resource classes, case attributes and resources to each other. Each task has one or more allocations. And an allocation may depend upon one or more case attributes. In most cases, an allocation will point to the intersection between a role and an organizational unit. In special cases, though, a specific resource may be excluded (separation of function) or selected (case manager).

3.2 ALLOCATION PRINCIPLES - AUTHORIZATION POLICY SPECIFICATION

The objective of a workflow system is to complete work items quickly and securely. In order to transform work items into activities, two decisions always need to be taken:

- Workflow Routing: In what order are the work items transformed into activities?
- Workflow Authorization: By which resource are the activities carried out?

It goes without saying that these two decisions are closely interrelated. The order can be important when selecting a resource. Conversely, the choice of a resource can affect the order in which work items are transformed into activities.

3.2.1 WORKFLOW ROUTING

A workflow can capture and manage several different types of events. Some of them are used for exception handling purposes, some for time-based synchronization of service execution, other for synchronization and dynamic data exchange with other services, and other again to monitor service state changes and react appropriately.

State change events

They are raised by changes in the process instance state. This includes the execution state of nodes and process instances as well as the value of process variables.

Error events

These are notifications of run-time errors occurred during process execution, such as when no service can be selected to execute a node.

Application-specific events

These events do not have a predefined semantics; they are qualified by their name, may carry several parameters, and are explicitly notified to the process engine by another process or by external parties. For instance, the customerCancel event.

Temporal events

They are raised at specified points in time. A workflow can detect and raise three types of temporal events:

- Instant: the event is raised only once, at a specified date and time.
- Periodic: these events are raised periodically.
- Interval: interval events are raised as a given interval has elapsed since a base event.

3.2.2 WORKFLOW AUTHORIZATION

In practice, the need arises for a flexible workflow authorization model, able to cope with the following requirements:

(i) **Instance authorizations**, to express authorizations valid only for specific workflow/task instances. Most WfMSs allow only the definition of authorizations at the workflow schema level. Thus, assignment criteria for tasks and cases are the same for all instances of the schema .While this approach is satisfactory for many workflows, sometimes additional authorizations need to be defined for a given case only, possibly depending on the value of workflow relevant data.

(ii) **Temporal authorizations**, to express authorizations with limited validity in time. Clearly, it is unpractical and unfeasible to have the workflow administrator modifying authorizations and prohibitions as the need arises, given the crucial role of authorization constraints and the huge number of workflow schemas and resources involved in constraint management. Hence, the model should provide flexible constructs in order to specify the time validity of authorizations.

(iii) **History authorizations**, to express the various authorization policies commonly applied in organizations for security, based on past actions in the execution of the activities.

The following categories of workflow authorization constraints are identified according to the building blocks of a workflow model:

A. Constraints on roles. These constraints impose restrictions on the executability of tasks by roles and agents. They allow specifying time-dependent and history-

dependent authorizations on agents and roles for their assignment to cases/tasks. Examples of this kind of constraints are the following:

1. Separation of duties: “Two different roles/agents must execute two tasks T1 and T2 in a given workflow W”.
2. Binding of duties: “The same role/agent must execute two tasks T1 and T2 in a given workflow W”.
3. Restricted task execution: “A role R can execute a task T in a given workflow W only for a given period of time”.
4. Restricted role membership: “An agent A can be assigned to a role R only for a given period of time”.

B. Constraints on cases/tasks. These constraints impose restrictions on the starting of cases and tasks. They allow specifying authorizations based on task and case properties or on time that must be satisfied for enabling the start event.
Examples of this kind of constraints are the following:

1. Case cooperation: “At the least K roles must be associated with a workflow W in order to start its execution”.
2. Task cooperation: “At the least K resources must be associated with a task T of a workflow W in order to start its execution”.
3. Inhibition: “A case/task is inhibited to all resources for a given period of time”.
4. Restricted activation: “After a number k of failed activations of a task T by the same resources A, A must be forbidden from the execution of T ”.

C. Constraints on workflow data. These constraints express authorizations on data accesses from within tasks, which depend on the nature (e.g., sensitive data) and the contents (e.g., data value) of data. Examples of constraints on data are the following:

1. Sensitive data filtering: “A (set of) field(s) of a sensitive information object O must be hidden to the task executor”.
2. Sensitive data management: “A task T must be assigned to a supervisor agent (e.g., the case responsible) when a sensitive information object is handled”.

Workflow authorization constraints can be further classified with respect to a set of criteria (Dependency, Scope and Time).

A. Dependency. We distinguish time, instance, and history dependency.

1. Time-dependency. This criterion captures the fact that a constraint can express authorizations having a temporal validity in workflow execution. We distinguish between time-dependent vs. time-independent authorization constraints. An example of time-dependent constraint is the restricted task execution while an example of time-independent constraint is the separation of duties.
2. Instance-dependency. This criterion captures the fact that a constraint expresses an authorization restricted to specific instances of a given workflow only, or spanning over all instances of a given workflow. We distinguish between instance-dependent vs. instance-independent authorization constraints. An example of instance-dependent constraint is the separation of duties while an example of instance-independent constraint is the case cooperation.
3. History-dependency. This criterion captures the fact that a constraint enforces an authorization dealing with past workflow activity execution, that we refer to as "history" (e.g., sequences of task activations, sequences of task assignments). We distinguish between history-dependent vs. history-independent authorization constraints. An example of history dependent constraint is the separation of duties while an example of history independent constraint is the restricted task execution.

B. Scope. We distinguish between global vs. local authorization constraints.

1. Local constraints express authorizations that hold in the scope of a given workflow schema only. All the examples of authorization constraints previously given are local.
2. Global constraints express an authorization which holds for all workflow schemas. An example of global authorization constraint is the following separation of duties: "The case responsible must be always different from any task executor".

C. Verification time. We distinguish between static vs. dynamic authorization constraints.

1. Static constraints are authorizations that should be checked at workflow compilation time. Case and task cooperation are examples of static constraints.
2. Dynamic constraints should instead be checked at workflow execution time, since they require the comparison between current and previous states of the workflow. Separation and binding of duties are examples of dynamic constraints.

3.2.3 CONTEXT SENSITIVE ACCESS CONTROL

Certain data access permissions of the workflow participants depend on the workflow execution context. In particular, contextual information available at access time, like temporal or user/customer relationship, can influence the authorization decision that allows a user to perform a task and access associated data objects. This enables a more flexible and precise authorization policy specification that incorporates the advantages of having broad, role-based permissions across workflow tasks and data object types, like RBAC, yet enhanced with the ability to simultaneously support the following features: (a) predicate-based access control, limiting user access to specific data objects, (b) a permission propagation function from one role holder to another in certain circumstances, (c) determining qualified task performers during a workflow instance based not only on the role-to-task permission policy, specified at workflow build time, but also on application data processed during the workflow instance, (d) location-based access control adapting user access to applications and data, to changing location contexts, (e) time-based access control, restricting user access to particular intervals or for a specified duration, depending on rules, by enabling and disabling user roles, (f) delegation-based access control, modifying access rights according to historical and runtime delegation qualifications.

Data content

Some role holders should be allowed to exercise a set of permissions on certain data objects only.

Permission propagation

Some role holders should receive additional to their roles permissions on certain data objects in order to effectively execute a task but these permissions should be revoked upon successful execution of the task.

Restricted task execution

In certain circumstances the candidates for a task instance execution should be dynamically determined and be either a sub-group of the authorized users or only one, specific authorized user.

Location Sensitive Access Control

The emerging, pervasive and ubiquitous computing environments need security services that are non-intrusive and easily adaptable to changing user or environmental contexts. Traditional security services are context insensitive. They require a complex and static authentication infrastructure in which a user has to identify himself by username and password or via certificates. In ad-hoc or roaming situations, access control could rather preferably depend on a group of users' contexts than on their identities. By coupling access control to such user or group context information, security services can become far more userfriendly and flexible.

The type of context information that is used for access control depends on the situation. It could be based on location, velocity, age, device and/or network capabilities, temperature, time of day, etc.

Time based Access Control

In many organizations, functions may have limited or periodic temporal duration. If part-time staff is represented by a role, then the above requirement entails that this role should be enabled only during the aforementioned temporal intervals. Similar requirements can be supported by specifying—for each role—the time periods in which they can be activated. We call role enabling the transition of a role from the nonenabled status to the enabled status, and role disabling the transition of a role from the enabled status to the nonenabled status. In a framework where roles are not always enabled, it is important to introduce ways to specify dependencies among the enabling and disabling of different roles.

Periodicity/duration constraints can be applied to roles themselves, by constraining the times when roles are enabled or active, and to user-role and role-permission assignments. User-role assignments refer to the permissions given to users to play roles, whereas role-permission assignments refer to the grants given to roles for accessing the protected resources. Role enabling as well as assignments can be restricted to particular intervals or for a specified duration depending on requirements.

Similarly, role activations can be restricted to duration constraints. Periodic expressions can be used to specify start and end times between which a role can be enabled, or an assignment granted. Duration constraints, however, only specify the length of time that a role can be enabled or activated, or an assignment granted, and do not specify start and end times, and hence such constraints come to effect only if some other events or conditions trigger them.

Delegation Conditions

There are three types of conditions that a user might want to make on delegating tasks to another authorized user:

1. Temporal Delegation Conditions - They allow a user to constrain delegation of a task to a defined time interval. This allows the delegator to set up a delegation to apply at some time in the future for some period of time as specified in the condition.
2. The time interval may also be periodic. Temporal conditions also allow volunteers to set time constraints on when they are willing to accept delegations. Finally, they also allow a potential delegatee to constrain rejection of tasks. As with delegation, the volunteer and delegatee can set up the constraints independent of the time in which these constraints will apply.
3. Workload Delegation Conditions - They allow a user to constrain delegation of a task to a workload level.
4. In other words, workload conditions allow the delegator to define a delegation that will only take place if assigned work- load exceeds a certain level. For the potential delegatee, it allows a rejection to be constrained by the workload that the potential delegatee already has. Finally, for volunteers, it al- lows them to control their commitment to accept delegations based on their workload level.
5. Value Delegation Conditions - They allow a user to constrain delegation of a task depending on attributes of the task. Therefore, a requirement of the workflow management system would be that workflow attributes be specified.

The above contextual constraint requirements will be the focus of our proposed workflow model which will be presented in the following sections.

Part II

Proposed Architecture

Part II contains the following chapters:

- Chapter 4, Tools
- Chapter 5, General Model Architecture

4

Tools

This chapter describes the tools and technologies that where used to develop the workflow model that will be presented in the following chapter.

This chapter includes the following sections:

- Section 4.1, " Web Services "
- Section 4.1.1, " The Great Promise Of Web Services "
- Section 4.1.2, " The Key Components "
- Section 4.1.3, " Advantages Of Web Services "
- Section 4.2, " Agents "
- Section 4.2.1, " What Is An Agent?"
- Section 4.2.2, " Jade And The Agents Paradigm "
- Section 4.2.3, " Jade Architecture "
- Section 4.3, " BPEL4WS "
- Section 4.3.1, " The Need For Business Processes "
- Section 4.3.2, " BPEL4WS For Service Composition "
- Section 4.3.3, " BPEL4WS Features "
- Section 4.3.4, " Relationship With WSDL "

4.1 WEB SERVICES

A Web service is a software application that can be accessed remotely using different XML-based languages. Normally, a Web service is identified by a URL, just like any other Web site. What makes Web services different from ordinary Web sites is the type of interaction that they can provide.

Most Web sites are designed to provide a response to a request from a person. The person either types in the URL of the site or clicks on a hyperlink to create the request. This request takes the form of a text document that contains some fairly

simple instructions for the server. These instructions are limited to the name of a document to be returned or a call to a server-side program, along with a few parameters.

A Web service is similar in that it is accessed via a URL. The difference lies in the content of what is sent in the request from the client to the service. Web service clients send an XML document formatted in a special way in accordance with the rules of the SOAP specification.

A SOAP message can contain a call to a method along with any parameters that might be needed. In addition, the message can contain a number of header items that further specify the intent of the client. These header items might designate what Web services will get this method call after the current service finishes its work, or they might contain security information. In any case, the complexity of the SOAP message far exceeds the complexity that is possible using only a browser.

4.1.1 THE GREAT PROMISE OF WEB SERVICES

Most of the enthusiasm surrounding Web services is based on the promise of interoperability. The Web services architecture is based on sending XML messages in a specific SOAP format. XML can be represented as plain ASCII characters, which can be transferred easily from computer to computer. The implications of this are significant:

- It doesn't matter what kind of computer sends the SOAP message or on what operating system it is running.
- It doesn't matter where in the world the client is sending the message from.
- It doesn't matter what language the software that the client is running on was written in.
- There is no need for the client to know what type of SOAP processor is running on the server.

In short, Web services are the Holy Grail of computing. Every software application in the world can potentially talk to every other software application in the world. This communication can take place across all the old boundaries of location, operating system, language, protocol, and so on.

4.1.2 THE KEY COMPONENTS

Web services transactions take place between components. For a Web service transaction to complete successfully, all of the components involved in processing the transaction must behave in ways that the other components expect them to. Given the differing vendors involved in the creation of the components, one might expect that they would have a lot of problems interacting. Although Web services interoperability is difficult to attain, it can be remedied by the creation of high-quality standards and a religious adherence to these standards by every programmer and vendor involved. The following are considered the core Web services standards:

- **SOAP**—SOAP originally stood for Simple Object Access Protocol. But SOAP is now considered a specification name and not an acronym. SOAP is a specification that defines an XML grammar for both sending messages and responding to messages that you receive from other parties. The goal of SOAP is to describe a message format that is not bound to any hardware or software architecture, but one that carries a message from any platform to any other platform in an unambiguous fashion.

  The SOAP standard contains two parts: the header that carries processing instructions and the body that contains the payload. The payload contains the information that you want to send. The two types of SOAP messages are documents and Remote Procedure Calls (RPCs). The payload of a document message is any XML document that you want moved from one computer to another. An RPC is a method call that is intended to be executed on the Web service's computer. The RPC message performs the same function as an ordinary method call in an ordinary programming language. The difference is that this call can take place over the Internet.

- **Extensible Markup Language (XML)**—Extensible Markup Language (XML) is the language that all the Web services Languages are built on. XML is a tool for constructing self-describing documents. In fact, XML is more of a meta-language than a language in that it is used to create grammars. These grammars are described in XML schemas that specify the tags that are allowed and the relationships between the elements defined by these tags. SOAP, WSDL, and UDDI are all XML-based grammars.

- **Hypertext Transport Protocol (HTTP)**—Hypertext Transport Protocol (HTTP) is a standard that precedes the advent of Web services. It was developed to facilitate the transfer of requests from a browser to a Web server. Web services takes advantage of the existence of this mature protocol to move SOAP messages and WSDL documents from one computer to another. Newer versions of SOAP describe how other transport mechanisms such as FTP, SMTP, and JMS can be used to perform this same function.

- **Web Services Description Language (WSDL)**—Web services Description Language (WSDL) is a specification that tells us how to describe a piece of software in terms of the method calls that it responds to. These methods are described in an abstract way that is independent of what programming language the actual service is written in or what computer and operating system it runs on. In fact, you can port an application from a personal computer to a mainframe, and the abstract portion of the WSDL description will not change (assuming that the port and protocol don't change when you port them). The WSDL also contains a concrete section in which the various details of how to actually make a connection to the service are stored. If a Web service could be accessed using HTTP, FTP, or SMTP, you would find three entries in the concrete section—one for each service.

- **Universal Discovery Description Integration (UDDI)**—The Universal Discovery, Description, and Integration (UDDI) specification describes how a potential customer of a Web service could learn about its capabilities and obtain the basic information needed to make the initial contact with the site. Normally, this contact includes a download of the WSDL.

  UDDI registries can be public, private, or semiprivate. A public directory allows everyone on the planet to examine the information that you post in the registry. A private registry exists behind the firewall of an organization and is only accessible by members of that organization. A semiprivate registry is open only to a limited number of outsiders such as best trading partners.

4.1.3 ADVANTAGES OF WEB SERVICES

There are many advantages to using the Web services architecture over any other. In fact, some Web services applications would be expensive or impossible to duplicate using any other technology. One of the challenges for technical leaders is to discover opportunities in which Web services can be used to solve today's problems. In addition, we need to be looking for Web services solutions to problems that are ever present, but never solved. These areas include the cost of doing business, in effect, the cost of software development, and the time that it takes to react to new market opportunities. The advantages that Web services provide over more traditional approaches are:

- Integrating legacy systems
- Lowering operational costs
- Lowering software development costs
- Getting systems done faster
- Interfacing with customers
- Integrating with external business partners
- Generating new revenue
- Supporting new business models

4.2 AGENTS

Agents are considered one of the most important paradigms that on the one hand may improve on current methods for conceptualizing, designing and implementing software systems, and on the other hand may be the solution to the legacy software integration problem.

4.2.1 WHAT IS AN AGENT?

The term 'agent', or software agent, has found its way into a number of technologies and has been widely used, for example, in artificial intelligence, databases, operating systems and computer networks literature. Although there is no single definition of an agent, all definitions agree that an agent is essentially a special software component that has autonomy that provides an interoperable interface to an arbitrary system and/or behaves like a human agent, working for some clients in pursuit of its own

agenda. Even if an agent system can be based on a solitary agent working within an environment and if necessary interacting with its users, usually they consist of multiple agents. These multi-agent systems (MAS) can model complex systems and introduce the possibility of agents having common or conflicting goals. These agents may interact with each other both indirectly (by acting on the environment) or directly (via communication and negotiation). Agents may decide to cooperate for mutual benefit or may compete to serve their own interests.

Therefore, an agent is autonomous, because it operates without the direct intervention of humans or others and has control over its actions and internal state. An agent is social, because it cooperates with humans or other agents in order to achieve its tasks. An agent is reactive, because it perceives its environment and responds in a timely fashion to changes that occur in the environment. And an agent is proactive, because it does not simply act in response to its environment but is able to exhibit goal-directed behaviour by taking initiative.

Moreover, if necessary an agent can be mobile, with the ability to travel between different nodes in a computer network. It can be truthful, providing the certainty that it will not deliberately communicate false information. It can be benevolent, always trying to perform what is asked of it. It can be rational, always acting in order to achieve its goals and never to prevent its goals being achieved, and it can learn, adapting itself to fit its environment and to the desires of its users.

4.2.2 JADE AND THE AGENTS PARADIGM

JADE is a software platform that provides basic middleware-layer functionalities which are independent of the specific application and which simplify the realization of distributed applications that exploit the software agent abstraction. A significant merit of JADE is that it implements this abstraction over a well-known object-oriented language, Java, providing a simple and friendly API. The following simple design choices were influenced by the agent abstraction.

An Agent is Autonomous and Proactive: An agent cannot provide call-backs or its own object reference to other agents in order to mitigate any chance of other entities coopting control of its services. An agent must have its own thread of execution,

using it to control its life cycle and decide autonomously when to perform which actions.

Agents Can Say 'No', and They are Loosely Coupled: Message-based asynchronous communication is the basic form of communication between agents in JADE; an agent wishing to communicate must send a message to an identified destination (or set of destinations). There is no temporal dependency between the sender and receivers: a receiver might not be available when the sender issues the message. There is also no need to obtain the object reference of receiver agents but just name identities that the message transport system is able to resolve into proper transport addresses. It is even possible that a precise receiver identity be unknown to the sender, which instead may define a receiver set using an intentional grouping or mediated by a proxy agent

Furthermore, this form of communication enables the receiver to select which messages to process and which to discard, as well as to define its own processing. It also enables the sender to control its thread of execution and thus not be blocked until the receiver processes the message. Finally, it also provides an interesting advantage when implementing multi-cast communication as an atomic operation rather than as N consecutive method calls (i.e. one send-type operation with a list of multiple message receivers instead of a method call for each remote object you wish to communicate with).

The System is Peer-to-Peer: Each agent is identified by a globally unique name (the AgentIdentifier, or AID, as defined by FIPA). It can join and leave a host platform at any time and can discover other agents through both white-page and yellow-page services (provided in JADE by AMS and the DF agents as defined also by the FIPA specifications). An agent can initiate communication with any other agent at any time it wishes and can equally be the object of an incoming communication at any time.
On the basis of these design choices, JADE was implemented to provide programmers with the following ready-to-use and easy-to-customize core functionalities.

- A fully distributed system inhabited by agents, each running as a separate thread, potentially on different remote machines, and capable of transparently communicating with one another, i.e. the platform provides a unique location-independent API that abstracts the underlying communication infrastructure.

- Full compliance with the FIPA specifications. The platform successfully participated in all FIPA interoperability events and was used as the middleware for many platforms in the Agentcities network (Agentcities). A great facilitator of this was active contribution by the JADE team to the FIPA standardization process.
- Efficient transport of asynchronous messages via a location-transparent API. The platform selects the best available means of communication and, when possible, avoids marshalling/unmarshalling java objects. When crossing platform boundaries, messages are automatically transformed from JADE's own internal Java representation into proper FIPA-compliant syntaxes, encodings and transport protocols.
- Implementations of both white pages and yellow pages. Federated systems can be implemented to represent domains and sub-domains as a graph of federated directories.
- A simple, yet effective, agent life-cycle management. When agents are created they are automatically assigned a globally unique identifier and a transport address which are used to register with their platform's white page service. Simple APIs and graphical tools are also provided to both locally and remotely manage agent life cycles, i.e. create, suspend, resume, freeze, thaw, migrate, clone and kill.
- Support for agent mobility. Both agent code and, under certain restrictions, agent state can migrate between processes and machines. Agent migration is made transparent to communicating agents that can continue to interact even during the migration process.
- A subscription mechanism for agents, and even external applications, that wish to subscribe with a platform to be notified of all platform events, including life-cycle-related events and message exchange events.
- A set of graphical tools to support programmers when debugging and monitoring. These are particularly important and complex in multi-threaded, multi-process, multi-machine systems such as a typical JADE application.
- Support for ontologies and content languages. Ontology checking and content encoding is performed automatically by the platform with programmers able to select preferred content languages and ontologies (e.g. XML and RDF-based). Programmers can also implement new content languages to fulfil specific application requirements.

- A library of interaction protocols which model typical patterns of communication oriented toward the achievement of one or more goal. Application-independent skeletons are available as a set of Java classes that can be customized with application-specific code. Interaction Protocols can also be represented and implemented as a set of concurrent finite state machines.
- Integration with various Web-based technologies including JSP, servlets, applets and Web service technology. The platform can also be easily configured to cross firewalls and use NAT systems.
- Support for J2ME platform and the wireless environment. The JADE run-time is available for the J2ME-CDC and -CLDC platforms through a uniform set of APIs covering both J2ME and J2SE environments.
- An in-process interface for launching/controlling a platform and its distributed components from an external application.
- An extensible kernel designed to allow programmers to extend platform functionality through the addition of kernel-level distributed services. This mechanism is inspired by the aspectoriented programming approach where different aspects can be woven into application code and coordinated at kernel level. To maintain compatibility with the J2ME environment which does not intrinsically support aspect-oriented code, JADE uses a special composition filter approach.

4.2.3 JADE ARCHITECTURE

A JADE platform is composed of agent containers that can be distributed over the network. Agents live in containers which are the Java process that provides the JADE run-time and all the services needed for hosting and executing agents. There is a special container, called the main container, which represents the bootstrap point of a platform: it is the first container to be launched and all other containers must join to a main container by registering with it.

The programmer identifies containers by simply using a logical name; by default the main container is named ‘Main Container’ while the others are named ‘Container-1’, ‘Container-2’, etc. Command-line options are available to override default names.

As a bootstrap point, the main container has the following special responsibilities:

- Managing the container table (CT), which is the registry of the object references and transport addresses of all container nodes composing the platform
- Managing the global agent descriptor table (GADT), which is the registry of all agents present in the platform, including their current status and location
- Hosting the AMS and the DF, the two special agents that provide the agent management and white page service, and the default yellow page service of the platform, respectively.

A common query is whether the main-container is a system bottleneck. In fact this is not the case as JADE provides a cache of the GADT that each container manages locally. Platform operations, in general, do not involve the main-container, but instead just the local cache and the two containers hosting the agents which are the subject and the object of the operation (e.g. sender and receiver of the message). When a container must discover where the recipient of a message lives, it first searches its LADT (local agent descriptor table) and then, only if the search fails, is the main container contacted in order to obtain the proper remote reference which, consequently, is cached locally for future usages. Because the system is dynamic (agents can migrate, terminate, or new agents can appear), occasionally the system may use a stale cached value resulting in an invalid address. In this case, the container receives an exception and is forced to refresh its cache against the main-container. The cache replacement policy is LRU (least recently used), which was designed to optimize long conversations rather than sporadic, single message exchange conversations which are actually fairly uncommon in multi-agent applications.

However, although the main-container is not a bottleneck, it does remain a single point of failure within the platform. To manage this, the platform exploits the Main Replication Service to ensure the JADE platform remains fully operational even in the event of a main-container failure. With this service the administrator can control the level of fault tolerance of the platform, the level of scalability of the platform, and somewhat uniquely, the level of distribution of the platform. A control layer composed of several distributed instances of the main-container can be configured to implement a distributed bootstrapping system (i.e. several bootstrap points can be passed to each container) and a distributed control system. In the extreme case, each container

can be instructed to join the Main Replication Service and act as part of the control layer.

Agent identity is contained within an Agent Identifier (AID), composed of a set of slots that comply with the structure and semantics defined by FIPA. The most basic elements of the AID are the agent name and its addresses. The name of an agent is a globally unique identifier that JADE constructs by concatenating a user-defined nickname (also known as a local name as it is sufficient for disambiguating intra-platform communication) to the platform name. The agent addresses are transport addresses inherited by the platform, where each platform address corresponds to an MTP (Message Transport Protocol) end point where FIPA-compliant messages can be sent and received.

Agent programmers are also allowed to add their own transport addresses to the AID when, for any application-specific purpose, they wish to implement their own agent private MTP.

When the main-container is launched, two special agents are automatically instantiated and started by JADE, whose roles are defined by the FIPA Agent Management standard:

1. The Agent Management System (AMS) is the agent that supervises the entire platform. It is the contact point for all agents that need to interact in order to access the white pages of the platform as well as to manage their life cycle. Every agent is required to register with the AMS (automatically carried out by JADE at agent start-up) in order to obtain a valid AID.
2. The Directory Facilitator (DF) is the agent that implements the yellow pages service, used by any agent wishing to register its services or search for other available services. The JADE DF also accepts subscriptions from agents that wish to be notified whenever a service registration or modification is made that matches some specified criteria. Multiple DFs can be started concurrently in order to distribute the yellow pages service across several domains. These DFs can be federated, if required, by establishing cross-registrations with one another which allow the propagation of agent requests across the entire federation.

4.3 BPEL4WS

4.3.1 THE NEED FOR BUSINESS PROCESSES

The goal of the Web Services effort is to achieve universal interoperability between applications by using Web standards. Web Services use a loosely coupled integration model to allow flexible integration of heterogeneous systems in a variety of domains including business-to-consumer, business-to-business and enterprise application integration. The following basic specifications originally defined the Web Services space: SOAP, Web Services Description Language (WSDL), and Universal Description, Discovery, and Integration (UDDI). SOAP defines an XML messaging protocol for basic service interoperability. WSDL introduces a common grammar for describing services. UDDI provides the infrastructure required to publish and discover services in a systematic way. Together, these specifications allow applications to find each other and interact following a loosely coupled, platform-independent model.

Systems integration requires more than the ability to conduct simple interactions by using standard protocols. The full potential of Web Services as an integration platform will be achieved only when applications and business processes are able to integrate their complex interactions by using a standard process integration model. The interaction model that is directly supported by WSDL is essentially a stateless model of synchronous or uncorrelated asynchronous interactions. Models for business interactions typically assume sequences of peer-to-peer message exchanges, both synchronous and asynchronous, within stateful, long-running interactions involving two or more parties. To define such business interactions, a formal description of the message exchange protocols used by business processes in their interactions is needed. The definition of such business protocols involves precisely specifying the mutually visible message exchange behaviour of each of the parties involved in the protocol, without revealing their internal implementation. There are two good reasons to separate the public aspects of business process behaviour from internal or private aspects. One is that businesses obviously do not want to reveal all their internal decision making and data management to their business partners. The other is that, even where this is not the case, separating public from private process provides the freedom to change private aspects of the process implementation without affecting the public business protocol.

Business protocols must clearly be described in a platform-independent manner and must capture all behavioural aspects that have cross-enterprise business significance. Each participant can then understand and plan for conformance to the business protocol without engaging in the process of human agreement that adds so much to the difficulty of establishing cross-enterprise automated business processes today.

What are the concepts required to describe business protocols? And what is the relationship of these concepts to those required to describe executable processes? To answer these questions, consider the following:

- Business protocols invariably include data-dependent behaviour. For example, a supply-chain protocol depends on data such as the number of line items in an order, the total value of an order, or a deliver-by deadline. Defining business intent in these cases requires the use of conditional and time-out constructs.
- The ability to specify exceptional conditions and their consequences, including recovery sequences, is at least as important for business protocols as the ability to define the behaviour in the "all goes well" case.
- Long-running interactions include multiple, often nested units of work, each with its own data requirements. Business protocols frequently require cross-partner coordination of the outcome (success or failure) of units of work at various levels of granularity.

If we wish to provide precise predictable descriptions of service behaviour for cross-enterprise business protocols, we need a rich process description notation with many features reminiscent of an executable language. The key distinction between public message exchange protocols and executable internal processes is that internal processes handle data in rich private ways that need not be described in public protocols.

In thinking about the data handling aspects of business protocols it is instructive to consider the analogy with network communication protocols. Network protocols define the shape and content of the protocol envelopes that flow on the wire, and the protocol behaviour they describe is driven solely by the data in these envelopes. In other words, there is a clear physical separation between protocol-relevant data and

"payload" data. The separation is far less clear cut in business protocols because the protocol-relevant data tends to be embedded in other application data.

4.3.2 BPEL4WS FOR SERVICE COMPOSITION

General adoption of business process automation solutions requires a standard foundation and a specialized language for composing services into business processes that provides the ability to express business processes in a standardized way, using a commonly accepted language. BPEL4WS is such a language and is quickly becoming the dominant standard. The main goal of BPEL4WS is to standardize the process of automation between web services.

Within enterprises, BPEL4WS is used to standardize enterprise application integration and extend the integration to previously isolated systems. Between enterprises, BPEL4WS enables easier and more effective integration with business partners. BPEL4WS stimulates enterprises to further define their business processes, which in turn leads to business process optimization, reengineering, and the selection of the most appropriate processes, thus further optimizing the organization. Definitions of business processes described in BPEL4WS do not influence existing systems. BPEL4WS is the key technology in environments where functionalities already are or will be exposed via web services. With increases in the use of web service technology, the importance of BPEL will rise further.

4.3.3 BPEL4WS FEATURES

With BPEL4WS we can define simple and complex business processes. To a certain extent, BPEL4WS is similar to traditional programming languages. It offers constructs, such as loops, branches, variables, assignments, etc. that allow us to define business processes in an algorithmic way. BPEL4WS is a specialized language focused on the definition of business processes. Therefore, on one hand it offers constructs, which make the definition of processes relatively simple. On the other hand, it is less complex than traditional programming languages, which simplifies learning.

The most important BPEL4WS constructs are related to the invocation of web services. BPEL4WS makes it easy to invoke operations of web services either synchronously or asynchronously. We can invoke operations either in sequence or in parallel. We can also wait for callbacks. BPEL4WS provides a rich vocabulary for fault handling, which is very important as robust business processes need to react to failures in a smart way. BPEL4WS also provides support for long-running process and compensation, which allows undoing partial work done by a process that has not finished successfully. Listed below are the most important features that BPEL4WS provides. With BPEL4WS we can:

- Describe the logic of business processes through composition of services
- Compose larger business processes out of smaller processes and services
- Handle synchronous and asynchronous (often long-running) operation invocations on services, and manage callbacks that occur at later times
- Invoke service operations in sequence or parallel
- Selectively compensate completed activities in case of failures
- Maintain multiple long-running transactional activities, which are also interruptible
- Resume interrupted or failed activities to minimize work to be redone
- Route incoming messages to the appropriate processes and activities
- Correlate requests within and across business processes
- Schedule activities based on the execution time and define their order of execution
- Execute activities in parallel and define how parallel flows merge based on synchronization conditions
- Structure business processes into several scopes
- Handle message-related and time-related events

BPEL4WS uses a notion of message properties to identify protocol-relevant data embedded in messages. Properties can be viewed as "transparent" data relevant to public aspects as opposed to the "opaque" data that internal/private functions use. Transparent data affects the public business protocol in a direct way, whereas opaque data is significant primarily to back-end systems and affects the business protocol only by creating nondeterminism because the way it affects decisions is

opaque. We take it as a principle that any data that is used to affect the behaviour of a business protocol must be transparent and hence viewed as a property.

The implicit effect of opaque data manifests itself through nondeterminism in the behaviour of services involved in business protocols. Consider the example of a purchasing protocol.

The seller has a service that receives a purchase order and responds with either acceptance or rejection based on a number of criteria, including availability of the goods and the credit of the buyer. Obviously, the decision processes are opaque, but the fact of the decision must be reflected as behaviour alternatives in the external business protocol. In other words, the protocol requires something like a switch activity in the behaviour of the seller's service but the selection of the branch taken is nondeterministic. Such nondeterminism can be modelled by allowing the assignment of a nondeterministic or opaque value to a message property, typically from an enumerated set of possibilities. The property can then be used in defining conditional behaviour that captures behavioural alternatives without revealing actual decision processes. BPEL4WS explicitly allows the use of nondeterministic data values to make it possible to capture the essence of public behaviour while hiding private aspects.

The basic concepts of BPEL4WS can be applied in one of two ways. A BPEL4WS process can define a business protocol role, using the notion of abstract process. For example, in a supply-chain protocol, the buyer and the seller are two distinct roles, each with its own abstract process. Their relationship is typically modelled as a partner link. Abstract processes use all the concepts of BPEL4WS but approach data handling in a way that reflects the level of abstraction required to describe public aspects of the business protocol. Specifically, abstract processes handle only protocol-relevant data. BPEL4WS provides a way to identify protocol-relevant data as message properties. In addition, abstract processes use nondeterministic data values to hide private aspects of behaviour.

It is also possible to use BPEL4WS to define an executable business process. The logic and state of the process determine the nature and sequence of the Web Service interactions conducted at each business partner, and thus the interaction protocols. While a BPEL4WS process definition is not required to be complete from a private implementation point of view, the language effectively defines a portable

execution format for business processes that rely exclusively on Web Service resources and XML data. Moreover, such processes execute and interact with their partners in a consistent way regardless of the supporting platform or programming model used by the implementation of the hosting environment.

Even where private implementation aspects use platform-dependent functionality, which is likely in many if not most realistic cases, the continuity of the basic conceptual model between abstract and executable processes in BPEL4WS makes it possible to export and import the public aspects embodied in business protocols as process or role templates while maintaining the intent and structure of the protocols. This is arguably the most attractive prospect for the use of BPEL4WS from the viewpoint of unlocking the potential of Web Services because it allows the development of tools and other technologies that greatly increase the level of automation and thereby lower the cost in establishing cross-enterprise automated business processes.

BPEL4WS defines a model and a grammar for describing the behaviour of a business process based on interactions between the process and its partners. The interaction with each partner occurs through Web Service interfaces, and the structure of the relationship at the interface level is encapsulated in what we call a partner link. The BPEL4WS process defines how multiple service interactions with these partners are coordinated to achieve a business goal, as well as the state and the logic necessary for this coordination. BPEL4WS also introduces systematic mechanisms for dealing with business exceptions and processing faults. Finally, BPEL4WS introduces a mechanism to define how individual or composite activities within a process are to be compensated in cases where exceptions occur or a partner requests reversal.

BPEL4WS is layered on top of several XML specifications: WSDL 1.1, XML Schema 1.0, and XPath1.0. WSDL messages and XML Schema type definitions provide the data model used by BPEL4WS processes. XPath provides support for data manipulation. All external resources and partners are represented as WSDL services. BPEL4WS provides extensibility to accommodate future versions of these standards, specifically the XPath and related standards used in XML computation.

4.3.4 RELATIONSHIP WITH WSDL

BPEL4WS depends on the following XML-based specifications: WSDL 1.1, XML Schema 1.0, XPath 1.0 and WS-Addressing.

Among these, WSDL has the most influence on the BPEL4WS language. The BPEL4WS process model is layered on top of the service model defined by WSDL 1.1. At the core of the BPEL4WS process model is the notion of peer-to-peer interaction between services described in WSDL; both the process and its partners are modeled as WSDL services. A business process defines how to coordinate the interactions between a process instance and its partners. In this sense, a BPEL4WS process definition provides and/or uses one or more WSDL services, and provides the description of the behavior and interactions of a process instance relative to its partners and resources through Web Service interfaces. That is, BPEL4WS defines the message exchange protocols followed by the business process of a specific role in the interaction.

The definition of a BPEL4WS business process also follows the WSDL model of separation between the abstract message contents used by the business process and deployment information (messages and portType versus binding and address information). In particular, a BPEL4WS process represents all partners and interactions with these partners in terms of abstract WSDL interfaces (portTypes and operations); no references are made to the actual services used by a process instance.

However, the abstract part of WSDL does not define the constraints imposed on the communication patterns supported by the concrete bindings. Therefore a BPEL4WS process may define behaviour relative to a partner service that is not supported by all possible bindings, and it may happen that some bindings are invalid for a BPEL4WS process definition.

A BPEL4WS process is a reusable definition that can be deployed in different ways and in different scenarios, while maintaining a uniform application-level behaviour across all of them.

5

Model Architecture

This chapter describes the architecture of the proposed workflow model and the problem it attempts to solve.

This chapter includes the following sections:

- Section 5.1, " What Does This Model Aim At?"
- Section 5.2, " Model Architecture "
- Section 5.3, " Implementation Issues "

5.1 WHAT DOES THIS MODEL AIM AT?

The model that is presented here, aims at simplifying security and workflow management of business processes. Security and workflow decisions concerning a business process are not only based on static rules and roles, but they are also determined by the context in which the process is executed. Context includes information about location, time, period of time, recourse availability, role activation and more.

Taking into account contextual information, enables a more flexible and precise authorization policy specification that incorporates the advantages of having broad, role-based permissions across workflow tasks and data object types, like RBAC, yet enhanced with the ability to simultaneously support the following features:

- Predicate-based access control, limiting user access to specific data objects
- A permission propagation function from one role holder to another in certain circumstances
- Determining qualified task performers during a workflow instance based not only on the role-to-task permission policy, specified at workflow build time, but also on application data processed during the workflow instance

- Location-based access control adapting user access to applications and data to changing user or environmental contexts
- Time-based access control restricting user access particular intervals or for a specified duration depending on requirements by enabling and disabling user roles
- Delegation-based access control, modifying access rights according to historical and runtime delegation qualifications

In order to create a flexible component-based architecture, that provides the ability to respond to increasingly complex business drivers and offer support to a diverse set of applications and service providers, use of a layered approach to constructing the model is warranted.

The presented model uses a multi-level architecture. Each level corresponds to a discrete functionality, the sum of which comprises a business process. We have distinguished six levels of functionality:

- Data Management
- Business functionality
- Process
- Security management
- Workflow management
- Presentation

5.2 MODEL ARCHITECTURE

Figure 1 presents the different levels of functionality, along with the data flow and the interactions between them:

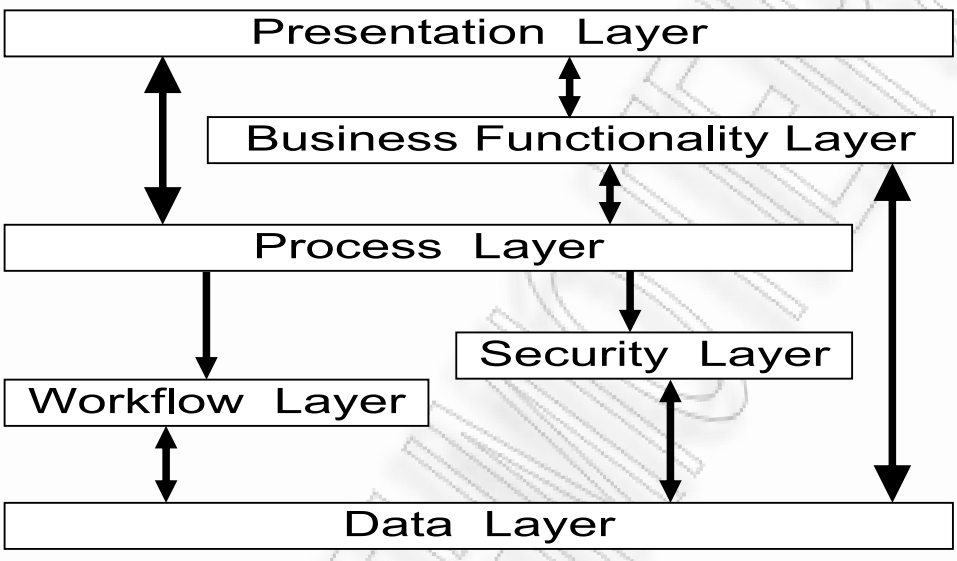


Figure1. Model Layers

Presentation Layer: It represents the business process interface. It is used by the process client to interact both with the workflow and the business functionality. Part of the client's actions result in altering the workflow state, while other actions aim at retrieving, updating or manipulating information through the use of the available web services.

Business Functionality Layer: It represents the applications that implement the business functionality. It consists of legacy systems, web services etc. It is invoked both by the workflow and the client and it results in the completion of a task. It interacts with the data layer, in order to gain access to all the necessary data.

Process Layer: It represents the business process in question and maps all its possible runtime scenarios. It involves a number of tasks, variables, conditions, alerts and events and interacts with all the other layers. For a process to be initiated, or to

change state, it is invoked by the client and it returns the results of the invoked operations to him. To accomplish a task, it invokes the business functionality layer. In order to capture the context of the process execution and to configure the access rights and the workflow routing, it invokes the security and the workflow management layers, passing the contextual information to them. It communicates with the data layer only through the layers it invokes.

Security Management Layer: This layer is responsible for the authorization control over data and task execution. It receives contextual information from the workflow process, retrieves the static security information from the data layer, applies the incorporated business rule logic to the aforementioned data, and updates the security database with the data and task execution access rights of the roles and clients involved in the specific instance of the business process.

Workflow Management Layer: This layer is responsible for the task assignment to the process resources. It receives contextual information from the workflow process, retrieves the static security information from the data layer along with the security information produced by the security layer, applies the incorporated business rule logic to the aforementioned data, and updates the security database with the task assignments of the clients involved in the specific instance of the business process.

Data Layer: It represents all the data involved in the business process. These include buldtime and runtime workflow data, security and business data. Its security content is analogous to the execution context of the process

5.3 IMPLEMENTATION ISSUES

In this section, we present the implementation of each level.
Figure 2 presents the deployment diagram of the proposed architecture.

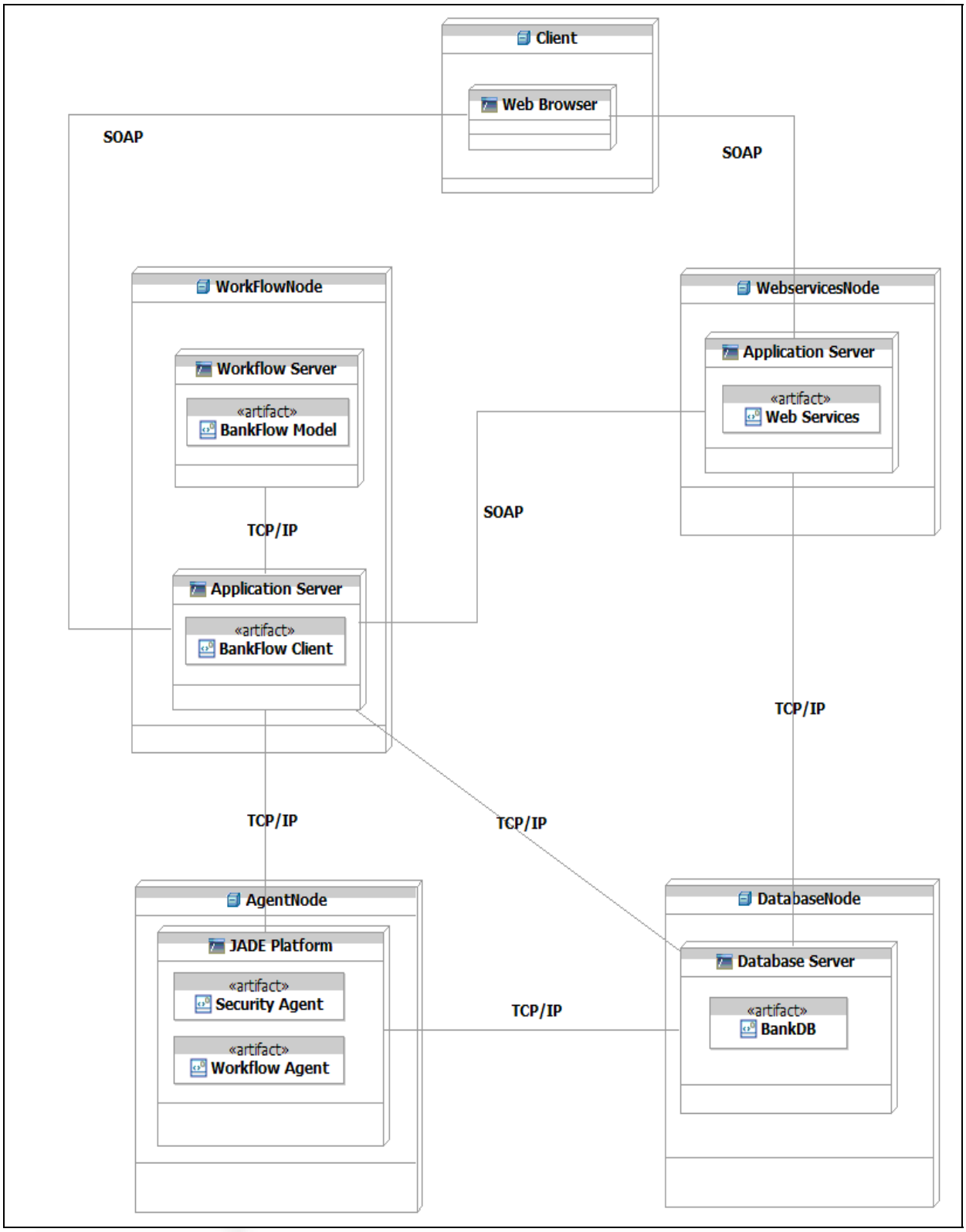


Figure2. Deployment Diagram

Presentation Layer: The client uses a web browser to communicate with the workflow and the business functionality. The interface is implemented in JSP.(Java Server Pages). Through a URL, the client accesses the main page of the interface and through a menu he can navigate to the available pages and thus the available functionality. The client can perform tasks that do not affect the state of the workflow. This means that not all functionality is attached to the workflow. The workflow is influenced only by a subset of the functionality. This makes sense, since there is no need to overburden the design of the workflow with operations that in essence do not affect the state of the process, such as informative operations.

Business Functionality Layer: This layer is implemented as a set of J2EE web services. In J2EE 1.4 platform, with its main focus on Web services, the existing Java-XML technologies are integrated into a consolidated platform in a standard way, thereby allowing applications to be exposed as Web services through a SOAP/HTTP interface. Web Services communicate with the BPEL process with the help of WSDL and Partner Links which are concrete references to services that a BPEL business process interacts with.

Process Layer: The process layer is the backbone of the presented model. It is implemented in BPEL4WS (Business Process Execution Language for Web Services). General adoption of business process automation solutions requires a standard foundation and a specialized language for composing services into business processes that provides the ability to express business processes in a standardized way, using a commonly accepted language. The main goal of BPEL is to standardize the process of automation between web services and legacy systems. Within enterprises, BPEL is used to standardize enterprise application integration and extend the integration to previously isolated systems. Definitions of business processes described in BPEL do not influence existing systems. BPEL is the key technology in environments where functionalities already are or will be exposed via web services. The process is accessible to the clients through WSDL, and it can be considered as a web service itself. The BPEL engine supports audit and instrumentation of workflow and thus is well suited for:

- Separating less volatile workflow steps from more volatile business rules
- Implementing line-of-business processes
- Implementing process flows requiring compensation

- Supporting large-scale instantiation of process flows
- Designing process flows that need auditing
- Orchestrating heterogeneous technologies such as connectors, web services, and Web Services Invocation Framework (WSIF)-enabled logic

Security Management Layer - Workflow Management Layer : Both security and workflow layers are implemented in JADE platform (Java Agent DEvelopment framework), which is used to build Multi-Agent systems. Each layer is implemented as an agent (Workflow Agent and Security Agent respectively). Agents are entirely implemented in JAVA. They execute tasks stimulated by an event, a message, or periodically. Te actual job, or jobs, an agent has to do is carried out within 'behaviours'. A behaviour represents a task that an agent can carry out and is implemented as an object of a class that extends jade.core.behaviours.Behaviour. To make an agent execute the task implemented by a behaviour object, the behaviour must be added to the agent. Behaviours can be added at any time when an agent starts up or from within other behaviours. An agent can execute several behaviours concurrently. Each behaviour corresponds to different agent functionality. When there are no behaviours available for execution the agent's thread goes to sleep in order not to consume CPU time. The thread is woken up again once a behaviour becomes available for execution again after a stimulus.

Agents communicate with other agents through a specific protocol based on asynchronous message passing. Thus, each agent has a 'mailbox' (the agent message queue) where the JADE run-time posts messages sent by other agents. Whenever a message is posted in the mailbox message queue the receiving agent is notified.

The communication with the workflow is accomplished in two steps

(a) WSIF (Web Services Invocation Framework) allows connectivity of BPEL processes with resources other than web services. It allows us to describe each service in WSDL, even if it is not a web service that communicates through SOAP. It also allows us to map such a service to the actual implementation and protocol. WSIF provides bindings for Java classes. To invoke any Java resource from BPEL we need to access data from BPEL variables sent as input parameters from Java, and send responses back to BPEL. Because BPEL variables are XML and Java

variables are not, we need a mapping between XML and Java. To handle XML data from Java we use XML façades. XML façades are a set of Java interfaces and classes through which we can access and modify XML data stored in BPEL variables using get/set methods.

(b) JADE agents can be started from an external application, and therefore the JADE run-time is created to host them. These external applications are JAVA classes that can communicate with the BPEL process with the help of WSIF. Data is transferred in the JAVA class, and an agent is created in order to send a message to the mailbox of the Security Agent or the Workflow Agent. Depending on the message, the right behaviour is executed.

Data Layer: This layer is implemented as two of databases used to store business, security and workflow data.

Figure 3 depicts the architecture of the proposed architecture.

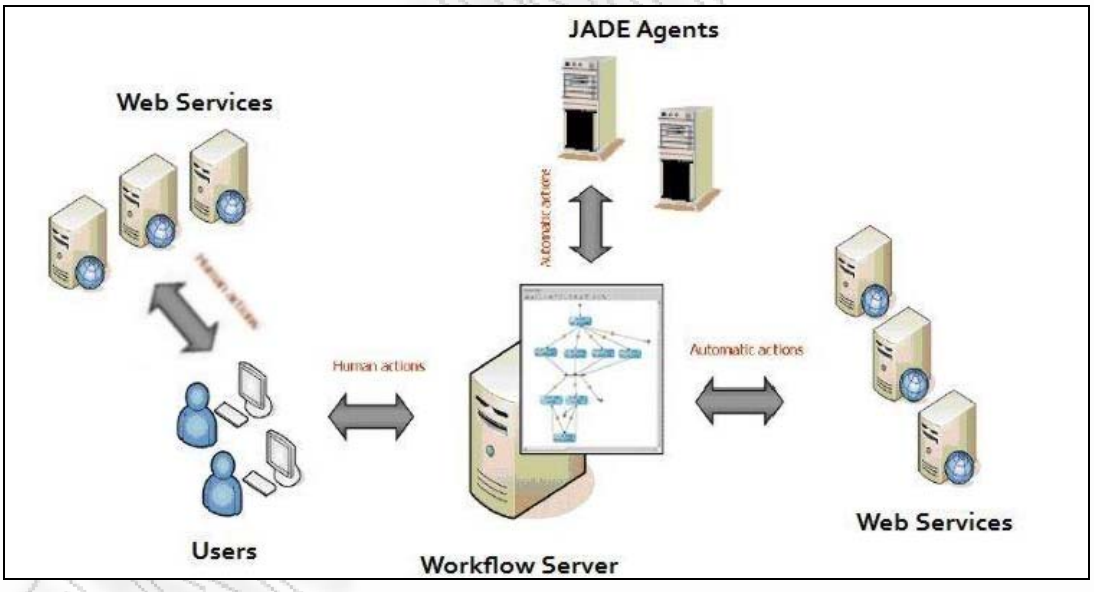


Figure 3. Model Architecture

Part III

Case Study

Part III contains the following chapters:

- Chapter 6, Case Study
- Chapter 7, Layer Implementation
- Chapter 8, Layer Communication Implementation
- Chapter 9, Runtime Demonstration

6

Problem Description

This chapter describes the case study that was implemented based on the proposed architecture.

This chapter includes the following sections:

- Section 6.1, " Case Study Scenario "
- Section 6.2, " Process Description "
- Section 6.3, " Database Description "

6.1 CASE STUDY SCENARIO

Suppose a customer portfolio management situation where an existing bank customer wishes to discuss restructuring his/her investment account portfolio with his/her personal customer relations officer (CRO) of the bank's investment accounts unit. If the CRO is unable to make a proposal that satisfies the customer on his/her own accord, he/she asks for an expert advice, regarding an appropriate product mix for the customer, from the market analysis unit of the bank. The request is sent to a specialist market analyst (MA) who, after assessing the customer's profile and conducting a market analysis, writes a consultation report and sends it to the requesting CRO.

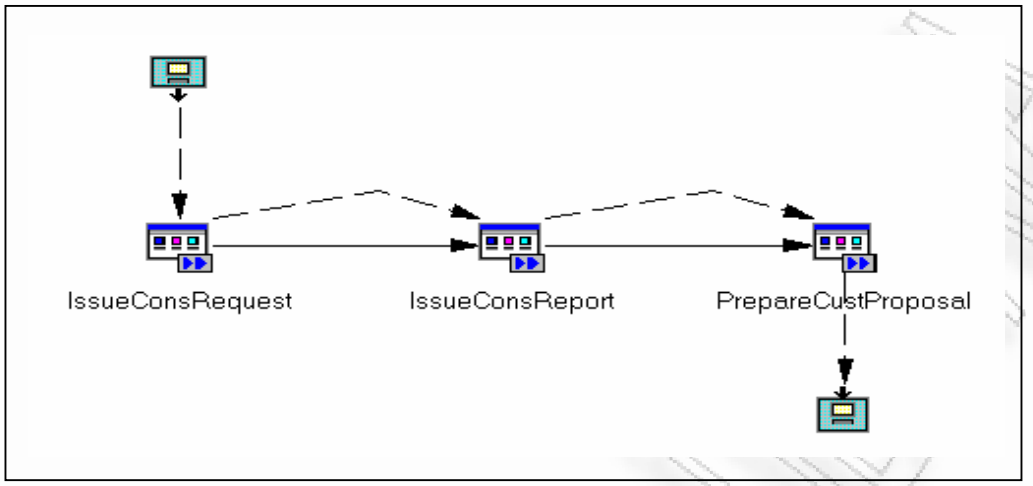


Figure 4: A high level business process model

Figure 4 shows a high-level view of a business sub-process concerned with customer portfolio management using the IBM WebSphere Workflow build-time tool. In this business process two organizational units of the bank are involved: the investment accounts unit and the market analysis unit. Two of the roles participating in the business process are: customer relations officer (CRO) and market analyst (MA). Table 1 shows an extract of workflow authorization requirements regarding task execution and related data access privileges assigned to these roles, respectively. Similar requirements exist in many workflow application fields, such as healthcare and insurance, where request-service business situations occur.

Table 1: Extract of authorization requirements for the business process of Figure 1 (Task execution and application data access permissions).

1.	CROs may issue consultation requests for their customers only (Issue_Consultation_Request).
1.1	CROs may write consultation requests for their current customers.
1.2	CROs may edit consultation requests for their current customers before sent.
1.3	CROs may send consultation requests for their current customers.
1.4	CROs may cancel consultation requests for their current customers after sent.
1.5	CROs may read customer records of their customers only.
2.	MAs may issue customer consultation reports on request by CROs (Issue_Consultation_Report).

2.1	MAs may read customer consultation requests issued by CROs
2.2	MAs may read customer records of customers they are requested to issue consultation reports for.
2.3	MAs may write customer consultation reports.
2.4	MAs may edit customer consultation reports before sent.
2.5	MAs may send customer consultation reports to the requesting CROs.
2.6	MAs may delegate assignments to other qualified MAs
2.7	MAs may cancel customer consultation reports after sent to the requesting CRO.
2.8	MAs may read past customer consultation reports prepared by them.
3.	CROs may prepare investment proposals for their customers after having received the consultation reports issued by MAs on request by them (Prepare_Customer_Proposal).
3.1	CROs may read the consultation reports issued by MAs on request by them.
3.2	CROs may write customer proposals for their current customers.
3.3	CROs may edit customer proposals for their current customers before presented to customers.
3.4	CROs may read customer records of their customers.
4.	CROs and MAs can access the system only from predefined locations
5.	CROs and MAs can access the system only on predefined time periods

6.2 PROCESS DESCRIPTION

A high-level view of the business process concerned with customer portfolio management is shown in Figure 5:

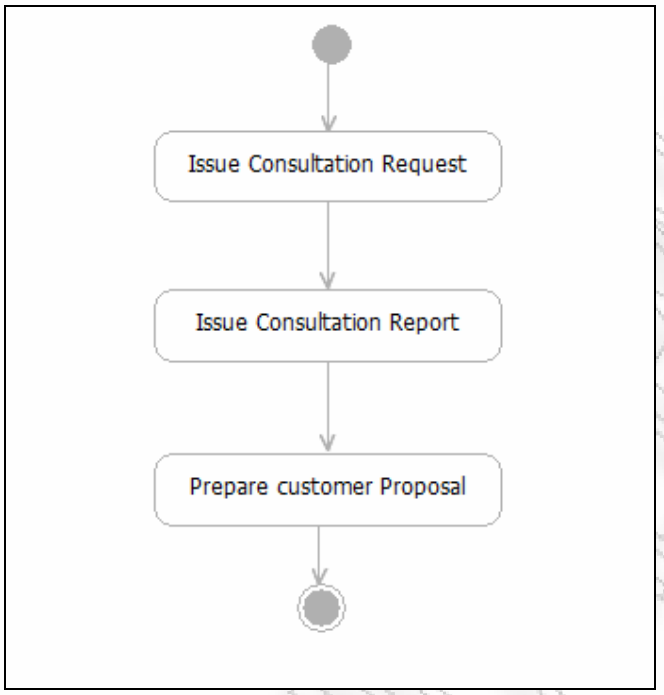


Figure 5. A high level business process model

Following, a more detailed description of the business process is presented:

- CRO logs in. Then:
    - The system checks whether the predefined from his department time period allows for his role to be activated
    - The system checks whether the location from which he logs in is acceptable (IP-address)
- CRO can access the information of his customers only
- CRO can issue a request for a given customer and specialty
- CRO can to edit the issued request
- CRO can send the request. Then:
    - A new process instance is created
    - The security agent (SA) defines the subgroup of MAs the request will be forwarded to based on their specialty.
    - The workflow agent (WA) consults the respective business rules and assigns the request to the MAs that rules evaluate to OK.

Business Rules:

1. RULE1 MAs who have declared that are going to be unavailable are checked. The ones, who are not available for 50%

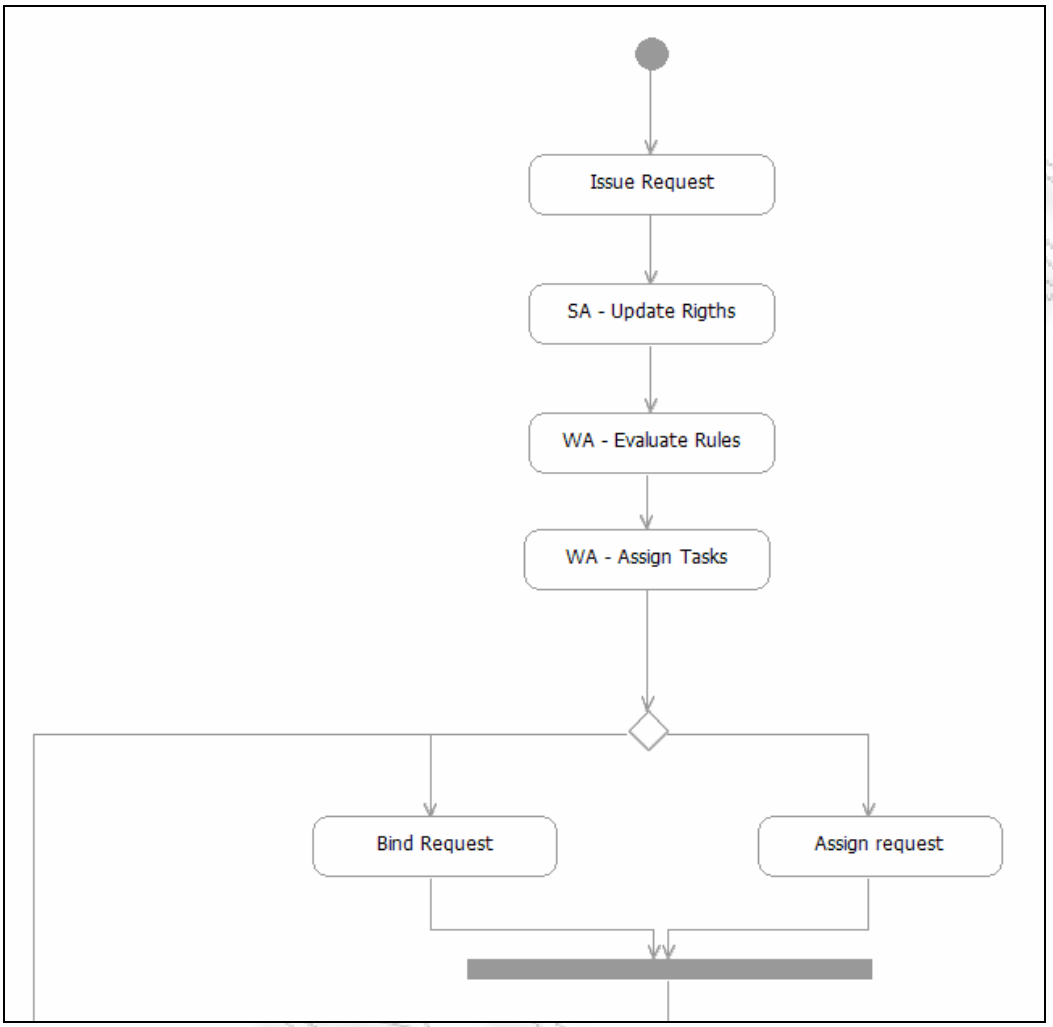
of the time period starting from time the request is issued and ending 10 days after, are excluded.

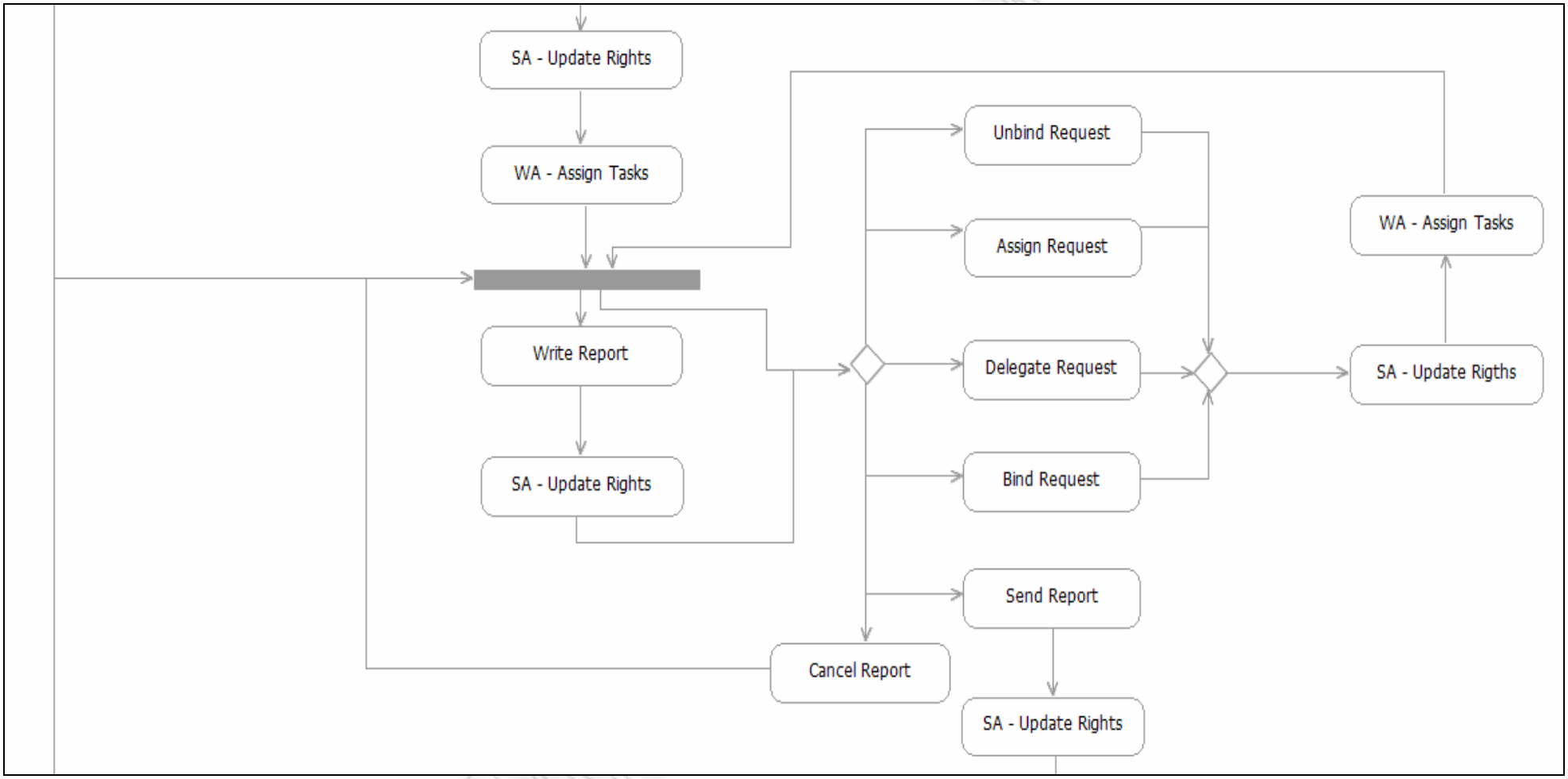
2. RULE2 Requests bound from the initially qualifying MAs are counted. The final qualifying list includes the 50% of the less burdened MAs.

- When the MAs search for new requests, a list with the requests that are assigned to them is returned.
- The MA who chooses to accept the request, binds it. Then:
    - WA removes the assignment from all other MAs
    - SA assigns to the MA that bound the request, access rights to the respective customer information, and execution rights to WRITE REPORT task.
- The MA decides to unbind the request. Then:
    - WA reassigns the request to all other initially qualifying MAs
    - SA revokes access rights to the respective customer information, and execution rights to WRITE REPORT task from the MA that unbound the request
- After a certain time elapses and the request is still unbound, WA informs the head of the Mas. In more detail:
    - Every day, time elapsed since the request was issued is evaluated for the unbound requests, and if it is greater than two days, the respective database table is updated.
    - The requests that are bound and not answered in a given timeline (10 days) are also evaluated and inserted in the respective database table.
- The head of the MAs searches for time critical requests and assignees the unbound request to one of the originally qualifying MAs. The state of that request, becomes urgent in the database table. When a request is urgent, the assigned MA can not unbind the request.
- At all times, the MA who bounded a request, can delegate it to another MA who is already included in the original qualifying list. Then:
    - SA and WA revoke MAs rights to access customer information and to execute the relevant tasks and assign the same access rights to the delegated MA
- The MA can write a report for a bound request. Then:

    - SA grants MA EDIT REPORT, READ REPORT, SEND REPORT task execution rights
- The MA can read a report for a bound request
- The MA can edit a report for a bound request
- The MA can send a report for a bound request. Then:
    - SA updates the state of the report, revokes MA access rights to the respective customer information and to the execution of relative tasks except READ REPORT task.
    - WA removes the qualifying MAs list, and places the report to the list of available reports to the CRO who has requested it
- The MA can only read a send report for a bound request
- The MA can cancel a send report. Then:
    - WA removes the report from the list of available reports to the CRO who has requested it
    - SA gives the respective data and task execution access rights to the MA
- CRO can search for new reports assigned to him
- CRO can read a report assigned to him
- CRO can write a proposal for a request he issued. Then:
    - SA assigns READ PROPOSAL, EDIT PROPOSAL and SEND PROPOSAL task execution rights to CRO
- CRO can edit a proposal
- CRO can delete a proposal
- CRO can send a proposal. Then:
    - SA updates the state of the proposal and removes all task execution access rights except READ PROPOSAL
- CRO can only read send proposals
- CRO can at all times, before sending a proposal, cancel the respective request. Then:
    - SA revokes all data and task execution access rights
    - MA removes all assignments for the given request

Figure 6 presents the detailed Activity Diagram for the above business process:


Issue Request
SA - Update Rigths
WA - Evaluate Rules
WA - Assign Tasks
Bind Request
Assign request


SA - Update Rights
WA - Assign Tasks
Write Report
SA - Update Rights
Unbind Request
Assign Request
Delegate Request
Bind Request
Send Report
Cancel Report
SA - Update Rights
WA - Assign Tasks
SA - Update Rigths

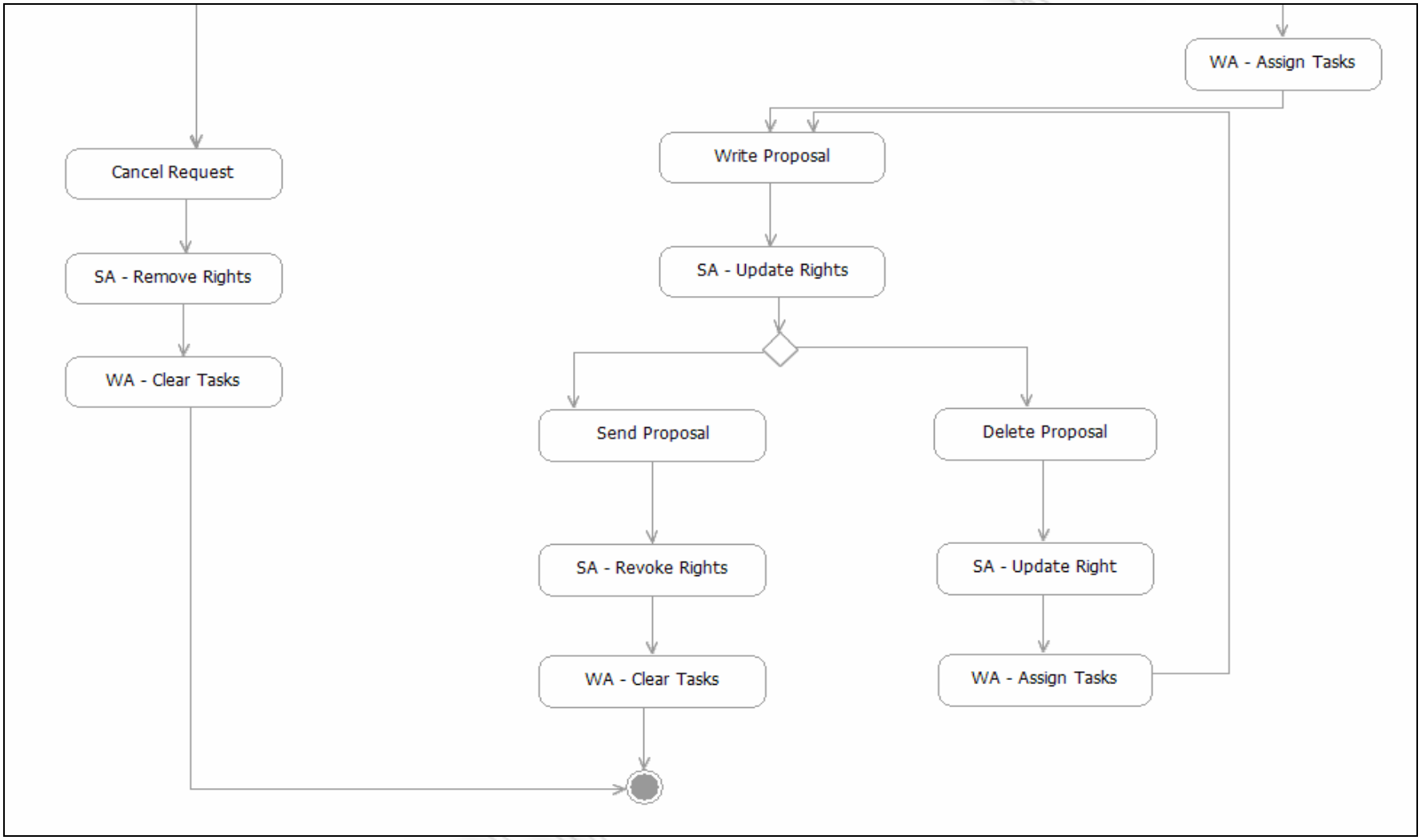


Figure 6: Activity Diagram

6.3 DATABASE DESCRIPTION

The model uses two databases. The first database stores The business relevant data, while the second database stores the security (data and task execution access rights) and workflow (task assignments) relevant data.

The business database consists of the following tables:

NAME	FIELDS	DESCRIPTION
account	(**customerid**, **number**, amount)	Stores information relevant to customer accounts
CRO	(**croid**, specid)	Stores CRO records
customer	(**customerid**, name, address, croid)	Stores customer records
employee	(**employeeid**, name, specid)	Stores employee records
MA	(maid, specid)	Stores MA records
portfolio	(**customerid**, product, datebought, datesold, pricebought, quantity)	Stores information relevant to customer portfolio
product	(**productid**, descry, price, spec)	Stores product records
proposal	(**requestid**, memo, state)	Stores proposal records
report	(**requestid**, maid, productid, quantity, pending)	Stores report details
request	(aa, **requestid**, croid, customerid, maid, dateplaced, pending, spec, convid)	Stores request details
spec	(**specid**, descr)	Stores specialty records

The security database consists of the following tables:

NAME	FIELDS	DESCRIPTION
accessperiod	(**employeeid**, roleid, timefrom, timeto, **datefrom**, **dateto**)	Defines the time period an employee's can be considered active
accessreport	(**employeeid**, **requestid**, mode)	Defines the employees task execution rights concerning the reports
assignreq	(**requestid**, **maid**, state)	Defines MAs request assignments
contrcust	(**customerid1**, **customerid2**)	Defines the contradicting customers
hotlistreq	(**requestid**, dateissued, state)	Defines the request that are time critical
macustomer	(**maid**, **customerid**)	MA assignment with access rights to customer information
location	(ip, **employeeid**, roleid)	Defines the location from where an employee can activate a role
notavail	(**maid**, datefrom, dateto)	Defines the time period when an employee is not available
request	(**requestid**, customerid, maid)	Stores request records
user	(**employeeid**, username, password)	Stores system user records

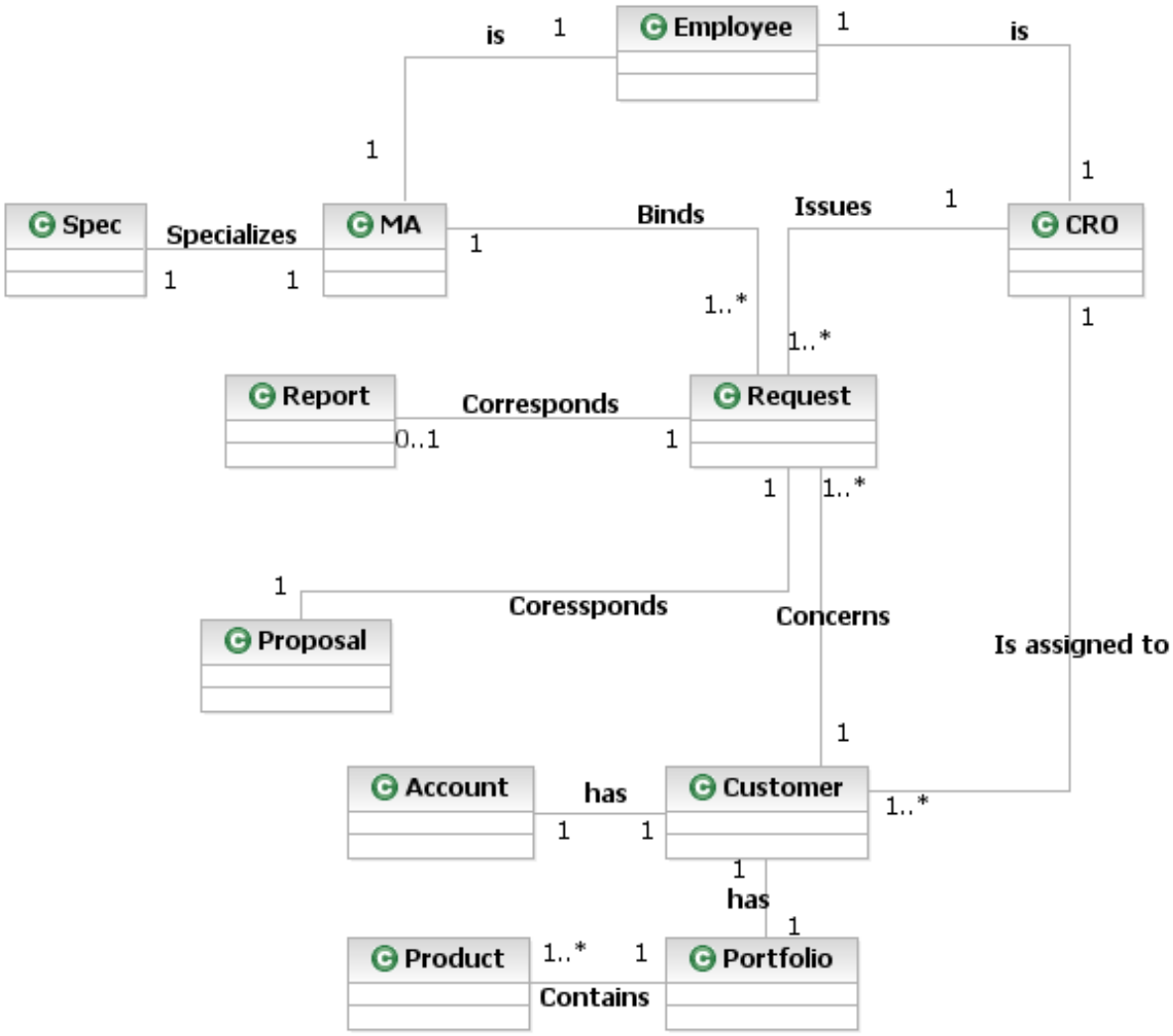


Figure 7. Business Database Class Diagram

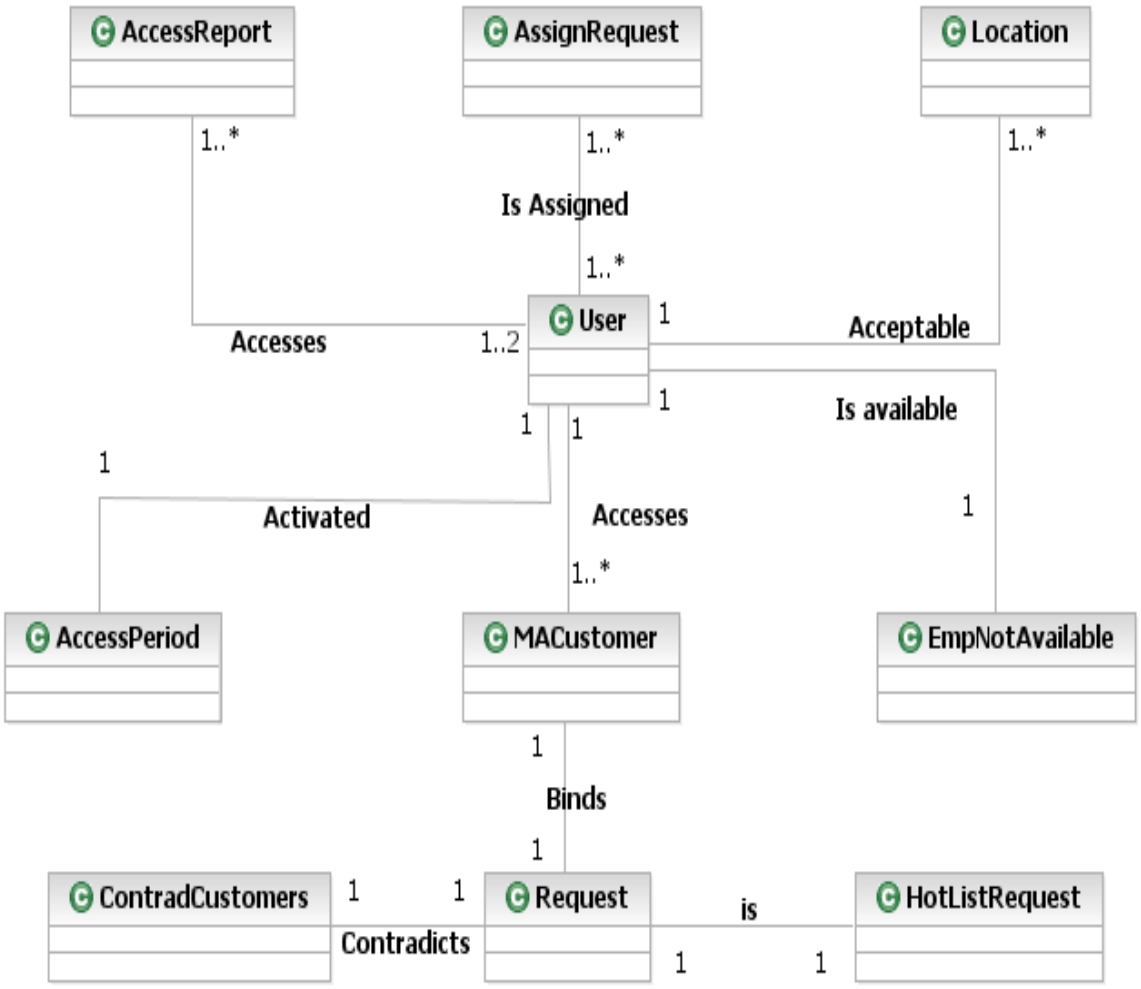


Figure 8. Security Database Class Diagram

6.4 SECURITY CONTEXT DESCRIPTION

Following, the security context, that derives from the above process is presented:

LOGIN. When a user tries to log in the system, three parameters define the security context: User identity, location and access time. So, three security requirements have to be met:

1. The identity of the user has to be confirmed through the username and password
2. The location of the user, which is defined through his IP, has to be valid
3. The time the user tries to enter the system has to comply with the rules set for him by his department

These requirements can be expressed as follows:

If ((login, username, password) AND (user_IP_Address in valid_ip_set) and (defined_access_time_from <= User_access_time<=defined_access_time_to)) then access=granted

CUSTOMER INFORMATION. A CRO has access only to the information of his customers. Two parameters define the security context: User identity and access time. Two security requirements have to be met in order for a CRO to read a customers' information:

1. The time the CRO tries to execute the task has to comply with the rules set for him by his department
2. The specific customer has to appear in the customer list the CRO can serve

These requirements can be expressed as follows:

If ((defined_access_time_from <= User_access_time<=defined_access_time_to) and (customer in (access, customer_set(CRO))))
then access=granted

ISSUE REQUEST. CRO can issue a request for a given customer and specialty. The same as the above two parameters define the security context and the same requirements have to be met:

1 The time the CRO tries to execute the task has to comply with the rules set for him by his department
2 The specific customer has to appear in the customer list the CRO can serve

These requirements can be expressed as follows:

If ((defined_access_time_from <= User_access_time<=defined_access_time_to) and (customer in (access, customer_set(CRO))))
then issue=granted

EDIT REQUEST. The CRO can edit a request he has issued. Two parameters define the security context: Access time and ownership. Two security requirements have to be met:

1 The time the CRO tries to execute the task has to comply with the rules set for him by his department
2 He has issued the request

These requirements can be expressed as follows:

If ((defined_access_time_from <= User_access_time<=defined_access_time_to) and (request in (issue, request_set(CRO))))
then edit=granted

SEND REQUEST. CRO can send a request he has issued. Again, two parameters define the security context: Access time and ownership.Two requirements have to be met:

1 The time the CRO tries to execute the task has to comply with the rules set for him by his department
2 CRO has issued the request

These requirements can be expressed as follows:

If ((defined_access_time_from <= User_access_time<=defined_access_time_to) and (request in (issue, request_set(CRO))))
then send=granted

When a CRO sends a request, a new process instance is created. The Security and Workflow Agents have to resolve the following security issues:

a. Which MAs will be allowed to handle the request?
b. Which is the subset of the above MAs that the business rules will allow the request to be forwarded to?

The following parameters define the security context: Access time, ownership, the speciality of the request and the predefined business rules which are evaluated against the following parameters: Current MA workload and current MA availability. So, the following requirements have to be met:

1 The MAs that have the request speciality are allowed to handle the request
2 The MAs that will be available for a certain number of days in the time period in which the request must be answered (e.g. not unavailable for more than five days for that time period), are initially qualified
3 The less burdened MAs are finally assigned with the request.

The first requirement, which is the Security Agents responsibility, can be expressed as follows:

If (MA, MA_speciality , request_spec) then MA in qualifying_MA_set

The last two requirements, which is the Workflow Agents responsibility, can be expressed as follows:

If ((MA in qualifying_MA_set) and (MA_availability>=(date_request_ answered – date_request_send-5))) then MA in qualifying_MA_subset

If ((MA in qualifying_MA_subset) and (MA_workload<=average(MA_set_workload))) then MA in final_qualifying_MA_subset

SEARCH REQUEST. When the MAs search for new requests, a list with the requests that are assigned to them is returned. Two parameters define the security context: Access time and ownership. Two security requirements have to be met:

1 The time the MA tries to execute the task has to comply with the rules set for him by his department
2 The MA must be assigned with a request by the Workflow Agent.

These requirements can be expressed as follows:

If (defined_access_time_from <= User_access_time<=defined_access_time_to)
then if(request in (assign, send_request_set(MA)))
then (request in MA_request_list)

BIND REQUEST. When an MA decides to serve a request, he binds it. This event reconfigures the security context and leads to the following actions:

a. WA removes the assignment from all other MAs by updating the state of the (MA, request) record in the respective security database table
b. SA assigns to the MA that bound the request, access rights to the respective customer information and execution rights to WRITE REPORT task by adding the records (MA, customer) and (MA, request, mode) in the respective security database table.

Three parameters define the security context: Access time, the request and the MA. Three security requirements have to be met:

1 Only a specific MA can access the request
2 The time the MA tries to execute the task has to comply with the rules set for him by his department
3 That MA has access rights to the respective customer information and execution rights to WRITE REPORT task.

The first requirement, which is the Workflow Agents responsibility, can be expressed as follows:

If (MA, bind , request) then access=granted else access =revoked

The two last requirements, which are the Security Agents responsibility, can be expressed as follows:

If ((defined_access_time_from <= User_access_time<=defined_access_time_to) and
(MA, access, request))
then ((MA, access, customer)=granted
and (MA, execute, request, write_report)=granted)

UNBIND REQUEST. If the MA decides to unbind the request, this event reconfigures the security context and leads to the following actions:

a. WA reassigns the request to all other initially qualifying MAs by updating the state of the (MA, request) pairs in the respective security database table
b. SA revokes access rights to the respective customer information, and execution rights to WRITE REPORT task from the MA that unbound the request by removing the records (MA, customer) and (MA, request, mode) from the respective security database table.

Three parameters define the security context: Access time, the request and the MA. three security requirements have to be met:

1 All the initially qualifying MAs can access the request
2 The time the MA tries to execute the task has to comply with the rules set for him by his department
3 No MA has access rights to the respective customer information and execution rights to WRITE REPORT task.

The first requirement, which is the Workflow Agents responsibility, can be expressed as follows:

If (MA, assign , request) then access=granted

The last two requirements, which are the Security Agents responsibility, can be expressed as follows:

If ((defined_access_time_from <= User_access_time<=defined_access_time_to) and
(MA, assign, request)) then ((MA, access, customer)=revoked
and (MA, execute, request, write_report)=revoked)

TIME CRITICAL REQUEST. After a certain time elapses and the request is still unbound, WA informs the head of the Mas. In more detail:

a. Every day, time elapsed since the request was issued is evaluated for the unbound requests, and if it is greater than two days, the respective database table is updated.
b. The requests that are bound and not answered in a given timeline (10 days) are also evaluated and inserted in the respective database table.

One parameter defines the security context: Time. So, one security requirement has to be met:

1 All delayed requests have to be identified and assigned to the head of MAs

This requirement, which is the Workflow Agents responsibility, can be expressed as follows:

If (request, date - date_request_send >5) then request in urgent_request_set

ASSIGN REQUEST. The head of the MAs searches for time critical requests and assignes the unbound request to one of the originally qualifying MAs.

Two parameters define the security context: The request and the initially qualifying MA list. Three security requirements have to be met:

1 Only the initially qualifying MAs can be assigned with the request
2 Only the MA that will be assigned with the request will be able to access it
3 That MA has access rights to the respective customer information and execution rights to WRITE REPORT task.

The first two requirements, which are the Workflow Agents responsibility, can be expressed as follows:

If (MA in final_qualifying_MA_subset) then assign=granted
If (MA, assign , request) then access=granted else access =revoked

The last requirement, which is the Security Agents responsibility, can be expressed as follows:

If (MA, access, request) then ((MA, access, customer)=granted
and (MA, execute, request, write_report)=granted)

DELEGATE REQUEST. At all times, the MA who bounded a request, can delegate it to another MA who is already included in the original qualifying list. This event reconfigures the security context and leads to the following actions:

a. SA and WA revoke MAs rights to access customer information and to execute the relevant tasks and assign the same access rights to the delegated MA

Four parameters define the security context: Access time, the request, the delegator and the degatee. Three security requirements have to be met:

1 Only the MA that will be delegated with the request will be able to access it
2 The time the MA tries to execute the task has to comply with the rules set for him by his department
3 The delegated MA has access rights to the respective customer information and execution rights to WRITE REPORT task.

The first requirement, which is the Workflow Agents responsibility, can be expressed as follows:

If (MA, delegate , request) then access=granted else access =revoked

The last two requirements, which are the Security Agents responsibility, can be expressed as follows:

If ((defined_access_time_from <= User_access_time<=defined_access_time_to) and
(MA, access, request))
then ((MA, access, customer)=granted
and (MA, execute, request, write_report)=granted)

WRITE REPORT. When a MA writes a report for a bound request, this event reconfigures the security context and leads to the following actions:

a. SA grants MA EDIT REPORT, READ REPORT, SEND REPORT task execution rights by inserting the records (MA, request, mode) to the respective security database table

Three parameters define the security context: Access time, the report and the MA. The following security requirements have to be met:

1 The time the MA tries to execute the task has to comply with the rules set for him by his department
2 The MA is granted EDIT REPORT, READ REPORT, SEND REPORT task execution rights

These requirements, which are the Security Agents responsibility, can be expressed as follows:

If ((defined_access_time_from <= User_access_time<=defined_access_time_to) and
(MA, write, report))
then ((MA, execute, report, edit_report)=granted
and (MA, execute, report, read_report)=granted
and (MA, execute, report, send_report)=granted)

READ REPORT. The MA can read a report for a bound request. Two parameters define the security context: Access time and ownership. Two security requirements have to be met:

1 The time the MA tries to execute the task has to comply with the rules set for him by his department
2 He has READ REPORT task execution rights

These requirements can be expressed as follows:

If ((defined_access_time_from <= User_access_time<=defined_access_time_to) and (MA, read,report))
then read=granted

EDIT REPORT. The MA can edit a report for a bound request. Two parameters define the security context:Access time and ownership. Two security requirements have to be met:

1 The time the MA tries to execute the task has to comply with the rules set for him by his department
2 He has EDIT REPORT task execution rights

These requirements can be expressed as follows:

If ((defined_access_time_from <= User_access_time<=defined_access_time_to) and (MA, edit,report))
then edit=granted

SEND REPORT. The MA can send a report for a bound request. This event reconfigures the security context and leads to the following actions:

a. SA updates the state of the report, revokes MA access rights to the respective customer information and to the execution of relative tasks except READ REPORT task by deleting the records (MA, customer) and (MA, request, mode) from the respective security database.
b. WA removes the qualifying MAs list, and places the report to the list of available reports to the CRO who has requested it by deleting the records (MA, request, state) and inserting the record (CRO, request, mode) to the respective security database tables.

Three parameters define the security context: Access time, the report and the MA. The following security requirements have to be met:

1 The time the MA tries to execute the task has to comply with the rules set for him by his department
2 The MAs access customer information rights and task execution rights are revoked, except READ REPORT task execution rights
3 The CRO is granted READ REPORT task execution rights
4 The qualifying MA list is deleted

The first three requirements, which are the Security Agents responsibility, can be expressed as follows:

If ((defined_access_time_from <= User_access_time<=defined_access_time_to)
then ((MA, execute, report, edit_report)=revoked
and (MA, execute, report, send_report)=revoked
and (MA, execute, report, delete_report)=revoked
and (CRO, execute, report, read_report)=granted)

The fourth requirement, which is the Workflow Agents responsibility, can be expressed as follows:

If (MA, assign, request) then assign=revoked

READ A SEND REPORT. The MA can only read a send report for a bound request. Two parameters define the security context: Time and ownership. Two security requirements have to be met:

1 The time the MA tries to execute the task has to comply with the rules set for him by his department
2 He has READ REPORT task execution rights

These requirements can be expressed as follows:

If ((defined_access_time_from <= User_access_time<=defined_access_time_to) and (MA, read,report))
then read=granted

CANCEL REPORT. The MA can cancel a send report. This event reconfigures the security context and leads to the following actions:

a. WA removes the report from the list of available reports to the CRO who has requested it by deleting the (CRO, report, mode) from the respective security database table
b. SA gives the respective data and task execution access rights to the MA by inserting the records (MA, customer) and (MA, request, mode) to the respective security database tables

Four parameters define the security context: Access time, the report, the CRO and the MA. The following security requirements have to be met:

1 The CROs READ REPORT task execution rights are revoked
2 The time the MA tries to execute the task has to comply with the rules set for him by his department
3 The MAs access customer information rights and task execution rights are granted

The first requirement, which is the Workflow Agents responsibility, can be expressed as follows:

If (CRO, read, report) then read=revoked

The last two requirements, which are the Security Agents responsibility, can be expressed as follows:

If ((defined_access_time_from <= User_access_time<=defined_access_time_to)
then ((MA, execute, report, write_report)=granted
and (MA, execute, report, read_report)=revoked
and (CRO, execute, report, read_report)=revoked)

SEARCH REPORT. CRO can search for new reports assigned to him. One parameter defines the security context: Access time. The following security requirement has to be met:

1 The time the CRO tries to execute the task has to comply with the rules set for him by his department

That requirement can be expressed as follows:

If ((defined_access_time_from <= User_access_time<=defined_access_time_to)
then search=granted

CRO READ REPORT. CRO can read a report assigned to him. Two parameters define the security context: Access time and ownership. Two security requirements have to be met:

1 The time the CRO tries to execute the task has to comply with the rules set for him by his department
2 He has READ REPORT task execution rights

These requirements can be expressed as follows:

If ((defined_access_time_from <= User_access_time<=defined_access_time_to) and (CRO, read,report))
then read=granted

WRITE PROPOSAL. CRO can write a proposal for a request he issued. this event reconfigures the security context and leads to the following actions:

a. SA grants CRO EDIT PROPOSAL, READ PROPOSAL, SEND PROPOSAL task execution rights by inserting the records (CRO, request, mode) to the respective security database table

Three parameters define the security context: Access time, the proposal and the CRO. The following security requirements have to be met:

1 The time the CRO tries to execute the task has to comply with the rules set for him by his department
2 The CRO is granted EDIT PROPOSAL, READ PROPOSAL, SEND PROPOSAL task execution rights

These requirements, which are the Security Agents responsibility, can be expressed as follows:

If ((defined_access_time_from <= User_access_time<=defined_access_time_to) and
(CRO, write, proposal))
then ((MA, execute, proposal, edit_ proposal)=granted
and (CRO, execute, proposal, read_ proposal)=granted
and (CRO, execute, proposal, send_ proposal)=granted)

EDIT PROPOSAL. CRO can edit a proposal. Two parameters define the security context: Access time and ownership. Two security requirements have to be met:

1 The time the CRO tries to execute the task has to comply with the rules set for him by his department
2 He has EDIT PROPOSAL task execution rights

These requirements can be expressed as follows:

If ((defined_access_time_from <= User_access_time<=defined_access_time_to) and (CRO, edit, proposal))
then edit=granted

DELETE PROPOSAL. CRO can delete a proposal. Two parameters define the security context: Access time and ownership. Two security requirements have to be met:

1 The time the CRO tries to execute the task has to comply with the rules set for him by his department
2 He has DELETE PROPOSAL task execution rights

These requirements can be expressed as follows:

If ((defined_access_time_from <= User_access_time<=defined_access_time_to) and (CRO, delete, proposal))
then delete=granted

SEND PROPOSAL. CRO can send a proposal. . This event reconfigures the security context and leads to the following actions:

a. SA updates the state of the request and revokes CROs execution rights of the relative tasks except READ PROPOSAL task by deleting the records (CRO, request, mode) from the respective security database.

Three parameters define the security context: Access time, the proposal and the CRO. The following security requirements have to be met:

1 The time the CRO tries to execute the task has to comply with the rules set for him by his department
2 The CROs task execution rights are revoked, except READ PROPOSAL task execution rights
3 The request state is updated to send.

These requirements, which are the Security Agents responsibility, can be expressed as follows:

If ((defined_access_time_from <= User_access_time<=defined_access_time_to) then ((CRO, execute, proposal, edit_report)=revoked
and (CRO, execute, proposal, delete_report)=revoked
and (CRO, execute, proposal, send_report)=revoked
and (request, send))

READ SEND PROPOSAL. CRO can only read send proposals. Two parameters define the security context: Access time and ownership. The following security requirements have to be met:

1 The time the CRO tries to execute the task has to comply with the rules set for him by his department
2 He has READ PROPOSAL task execution rights

These requirements can be expressed as follows:

If ((defined_access_time_from <= User_access_time<=defined_access_time_to) and (CRO, read, proposal))
then read=granted

CANCEL REQUEST. CRO can at all times, before sending a proposal, cancel the respective request. . This event reconfigures the security context and leads to the following actions:

a. SA revokes all data and task execution access rights by deleting the respective records from the security database tables
b. WA removes all assignments for the given request by deleting the respective records from the security database tables

Two parameters define the security context: Access time, the request. The following security requirements have to be met:

1 The time the CRO tries to execute the task has to comply with the rules set for him by his department
2 All the respective access and task execution rights have to be removed
3 All the respective assignments have to be removed

The first two requirements, which are the Security Agents responsibility, can be expressed as follows:

If ((defined_access_time_from <= User_access_time<=defined_access_time_to) then ((CRO, execute, request)=revoked
and (MA, execute, request)=revoked
and (MA, customer)=revoked)

The third requirement, which is the Workflow Agents responsibility, can be expressed as follows:

If (MA, assign, request) then assign=revoked

7

Layer Implementation

This chapter describes the implementation of the Model Layers.

This chapter includes the following sections:

- Section 7.1, " Business Functionality Layer "
- Section 7.1.1, " Web Service Operations "
- Section 7.1.1.1, " User Authentication Operations "
- Section 7.1.1.2, " Information Retrieval Operations "
- Section 7.1.1.3, " Request Management Operations "
- Section 7.1.1.4, " Report Management Operations "
- Section 7.1.1.5, " Proposal Management Operations "
- Section 7.2, " Security Layer "
- Section 7.3, " Workflow Layer "
- Section 7.4, " Business Process Layer "

7.1 BUSINESS FUNCTIONALITY LAYER

This section describes the bussines functionality that is implemented in the Business Functionality Layer.

7.1.1 WEB SERVICE OPERATIONS

The business functionality layer in implemented as a set of orchestrated web services. Its functionality consists of the following:

- User authentication operations
- Information retrieval operations
- Request management operations
- Report management operations
- Proposal management operations

Java Beans are used to transfer data between the databases, the web services and the client.

The java beans that are used are the following:

- AccountBean
- CustomerBean
- MaBeanPortfolioBean
- ProductBean
- ProposalBean
- ReportBean
- RequestBean

7.1.1.1 USER AUTHENTICATION OPERATIONS

The user authentication operations are the following:

- **Login-**Clients declare their identity by a username and a password. These two values are transferred to the web service login(String username, String password) where they are checked against the data stored in the database table User(username, password, employeeid). If there exists a record for that pair of values, the employeeid is returned.
- **Login_IP**- Clients declare their location by their IP-address. An employee in allowed to access the system only after the location he is in, is checked against information stored in the database table location (ip, employeeid, roleid). The system detects the clients IP address and along with his employeeid it transfers them to the web service login_IP(String IP, String employeeid). If there exists a record for that pair of values, the employee is granted access to the system.
- **Login_Time** - Clients are checked against the time constraints from the security database, when they try to access the system. The time the client logs in the system is compared to the data in the table accessperiod (employeeid, roleid, timefrom, timeto, datefrom, dateto). The time periods the specific client has the permission from his department to access the system are stored in that table. The clients employeeid and access time are transferred to the web service login_time(String time, String employeeid). If there exists a record for that pair of values, the employee is granted access to the system.

  This service can be called each time a client logs in the system or each time he tries to execute a task.

7.1.1.2 INFORMATION RETRIEVAL OPERATIONS

CROs and MAs who are given permission to access customer data, can do so using the following web services:

- **Customer List** - The list of customers a CRO or a MA has information access to, is provided by the web service search_employee_cust (java.lang.String croid, java.lang.String customerid) which returns an array of Customerbeans.
- **Customer Record** - The record of a customer that is handled by a specific CRO is returned by the web service search_cust(java.lang.String croid, java.lang.String customerid) in the form of CustomerBean.
- **MA Customer List** - The record of a customer that is handled by a specific MA is returned by the web service search_ma_employee_cust (java.lang.String maid, java.lang.String customerid) in the form of CustomerBean.
- **Account Record** - The information about the accounts of a customer is returned from the web service search_account(java.lang.String customerid) as an array of AccountBeans.
- **Portfolio Record** - The information about the portfolio of a customer is returned from the web service search_portfolio (java.lang.String customerid) as an array of PortfolioBeans.
- **Product Record** - The information about the available products is returned from the web service search_ product (java.lang.String employeeid) as an array of ProductBeans.

7.1.1.3 REQUEST MANAGEMENT OPERATIONS

The request management operations are the following:

- **Issue Request** - When a CRO decides to issue a request for one of the customers that are assigned to him, he fills a form with information, specifying the details and then calls the web service issuerequest(String id, String customerid, String spec, String convid) in order to save the request.
- **Edit Request** - If a CRO wishes to edit or send a request he has previously issued, he searches the database and receives a list of all his pending requests. To accomplish this, he calls the service cro_search_request(java.lang.String croid) which returns an array of RequestBeans.

- **Update Request** - If a CRO decides to alter an issued and pending request, he calls the service update_request(String reqid, String spec) which gets the client changes and updates the respective database table.
- **Send Request** - When a CRO decides to send a pending request to the MA department, he calls the service sendrequest(java.lang.String requestid), which alters the state of the request.
- **ConversationID** - For each request that is send, a new instance of the bussines process is created. Every instance has a unique identifier, the conversationId, which is stored in the request database table. Every time a client wants to interact with the process for a specific request, has to recover the ConversationId that corresponds to the request in question. This in achived through the service convid_request(String requested), which returns the conversationId of the request.
- **Search Request** - When a MA wants to search for the requests that are assigned to him, he calls the service search_request(java.lang.String maid) which returns an array of RequestBeans.
- **Bind Request** - If a MA decides to bind a request, he calls the service bindrequest(java.lang.String maid, java.lang.String requestid), which alters the state of the request and updates the respective database table with the MAs id.
- **Unbind Request** - If a MA decides to unbind a request, he calls the service unbindrequest (java.lang.String maid, java.lang.String requestid), which alters the state of the request and deletes from the respective database table the MAs id.
- **Assign Request** - When a request is not bound a certain time after it has been send, its state is updated to hot by the workflow agent, and the request appears in the list of time critical requests. The head of MAs checks periodically this list and assigns the requests that appear in it, by calling the service assignrequest(String maid, String requestid). This service alters the state of the request and updates the respective database table with the MAs id.
- **Hot Request List** - The list of time critical requests is returned by the service search_hl_request (java.lang.String maid) as an array of RequestBeans.
- **Delegate Request** - An MA that has decided to bind a request, can delegate it to one of the other MAs who where initially assigned with that request. The delegation takes place with the use of the service delegatereq(String maidfrom, String maidto, String requestid) that updates the respective database table with the new MAs id.
- **Delegatee List** - The list of the qualifying for the delegation MAs is returned by the service list_deleg(java.lang.String requestid) as an array of MABeans.

- **MAs Request List** - The list of a MAs bound requests is returned be the service search_b_request (java.lang.String maid) as an array of RequestBeans.

7.1.1.4 REPORT MANAGEMENT OPERATIONS

The report management operations are the following:

- **Add Report** - When a MA decides to write a report for a request he has bounded, he fills a form with relevant information and calls the web service add_report(String reqid, String maid, String prodid, String quantity) in order to save the report record. He will have to call the same service repeatedly to add more records to the same report.
- **Search Report** - When a MA wishes to search for the reports he has written, he calls the service search_report (java.lang.String maid), which returns the list of the MAs reports as an array of ReportBeans.
- **Delete Report** - When an MA decides to delete a specific record from a report, he calls the service delete_report(String reqid, String product), which removes the record from the respective database table.
- **Cancel Report** - When an MA decides to cancel a report, he calls the service delete_all_report(String reqid), which removes all the records of that report from the respective database table.
- **Edit Report** - If a MA decides to alter a written and pending report, he calls the service update_report(String reqid, String product, String quantity) which reads the client changes and updates the respective database table.
- **Send Report** - When a MA decides to send a pending report to the CRO that has issued the respective request, he calls the service send_report (java.lang.String requestid), which alters the state of the report and the respective request.

7.1.1.5 PROPOSAL MANAGEMENT OPERATIONS

The proposal management operations are the following:

- **Write Proposal** - When a CRO decides to write a proposal based on one of the reports he has received for a request he has issued, he fills a form with relevant information and the proposal text and calls the web service write_proposal (String reqid, String memo) in order to save the proposal.

- **Search Proposal** - When a CRO wishes to search for the proposals he has written, he calls the service read_all_proposal(java.lang.String croid), which returns the list of the CROs proposals as an array of ProposalBeans.
- **Edit Proposal** - If a CRO decides to alter a written and pending proposal, he calls the service update_proposal(String reqid, String memo) which reads the client changes and updates the respective database table.
- **Delete Proposal** - If a CRO decides to delete a written and pending proposal, he calls the service delete_proposal(String reqid) which deletes the proposal from the respective database table.
- **Send Proposal** - When a CRO decides to send a pending proposal to his customer, he calls the service send_proposal (java.lang.String requestid), which alters the state of the proposal and the respective request.
- **Read Proposal** - When a CRO decides to read a proposal, he calls the service read_proposal (java.lang.String requestid), which retrieves the proposal from the database and returns it to the CRO.

The code of the above mentioned web services along with the corresponding WSDL file are presented in Apendix A.

7.2 SECURITY LAYER

The security layer is implemented in JADE platform. A JADE agent accepts messages from the BPEL process, that include information about the workflow runtime context, and adjusts accordingly the data and task execution access rights. Whenever a web service that calls for access rights reconfiguration is executed, the security agent takes over, to reflect the new context in the security database.

Its functionality is implemented by ten cyclic behaviors. 'Cyclic' behaviours are designed to never complete; their action() method executes the same operations each time it is called. The security agent is used in the following situations:

- When a CRO issues a new request
- When a MA binds a request
- When a MA unbinds a request
- When a MA delegates a request
- When a MA writes a new report
- When a MA sends a report
- When a MA cancels a report
- When a CRO writes a new proposal
- When a CRO sends a proposal
- When a CRO cancels a request

All other actions do not influence the process context and thus the security requirements and the security agent is not activated.

Following, the messages received by the Security Agent from the workflow and the Security Agent actions are described in more detail:

SEND REQUEST - When a new request is issued, the security agent receives a message that includes the requestid and the request specialty. It then updates the security database. It retrieves all the MAs of the requested specialty, and inserts the records (maid, requestid,state) in the respective database table.

Two parameters define the security context: Access time and ownership. Two security requirements have to be met:

1 The time the CRO tries to execute the task has to comply with the rules set for him by his department
2 He has issued the request

These requirements can be expressed as follows:

If ((defined_access_time_from <= User_access_time<=defined_access_time_to) and (request in (issue, request_set(CRO))))
then send=granted

When a CRO sends a request, a new process instance is created. The Security and Workflow Agents have to resolve the following security issues:

a. Which MAs will be allowed to handle the request?
b. Which is the subset of the above MAs that the business rules will allow the request to be forwarded to?

The following parameters define the security context: Access time, ownership, the speciality of the request and the predefined business rules which are evaluated against the following parameters: Current MA workload and current MA availability. So, the following requirements have to be met:

1 The MAs that have the request speciality are allowed to handle the request
2 The MAs that will be available for a certain number of days in the time period in which the request must be answered (e.g. not unavailable for more than five days for that time period), are initially qualified
3 The less burdened MAs are finally assigned with the request.

The first requirement, which is the Security Agents responsibility, can be expressed as follows:

If (MA, speciality , request_spec) then MA in qualifying_MA_set

BIND REQUEST - When a MA binds a request, the security agent receives a message that includes the requestid, the maid, and the customerid. It then updates the security database. It grants the MA access to the relevant information of the customer the request is issued for. It also grants the MA the right to execute the WRITE REPORT task for the particular request.

Three parameters define the security context: Access time, the request and the MA. The following security requirements have to be met:

1 Only a specific MA can access the request
2 The time the MA tries to execute the task has to comply with the rules set for him by his department
3 That MA has access rights to the respective customer information and execution rights to WRITE REPORT task.

The last two requirements, which are the Security Agents responsibility, can be expressed as follows:

If ((defined_access_time_from <= User_access_time<=defined_access_time_to) and
(MA, access, request))
then ((MA, access, customer)=granted
and (MA, execute, request, write_report)=granted)

UNBIND REQUEST - When a MA unbinds a request, the security agent receives a message that includes the requestid, the maid and the customerid. It then updates the security database. It revokes the MAs access rights to the relevant information of the customer the request is issued for. It also revokes the MAs right to execute ant type of task (white, read, edit) for the particular request.

Three parameters define the security context: Access time, the request and the MA. So, two security requirements have to be met:

1 All the initially qualifying MAs can access the request
2 The time the MA tries to execute the task has to comply with the rules set for him by his department
3 No MA has access rights to the respective customer information and execution rights to WRITE REPORT task.

The last two requirements, which are the Security Agents responsibility, can be expressed as follows:

If ((defined_access_time_from <= User_access_time<=defined_access_time_to) and
(MA, assign, request)) then ((MA, access, customer)=revoked

and (MA, execute, request, write_report)=revoked)

DELEGATE REQUEST - When a MA decides to delegate a request, the security agent receives a message that includes the requestid, the maid, a second maid of the MA the request will be delegated to and the customerid. It then updates the security database. It revokes the MAs access rights to the relevant information of the customer the request is issued for. It also revokes the MAs right to execute ant type of task (white, read, edit) for the particular request and grants the same rights to the second MA.

Four parameters define the security context: Access time, the request, the delegator and the degatee. Three security requirements have to be met:

1 Only the MA that will be delegated with the request will be able to access it
2 The time the MA tries to execute the task has to comply with the rules set for him by his department
3 That MA has access rights to the respective customer information and execution rights to WRITE REPORT task.

The last two requirements, which are the Security Agents responsibility, can be expressed as follows:

If ((defined_access_time_from <= User_access_time<=defined_access_time_to) and
(MA, access, request))
then ((MA, access, customer)=granted
and (MA, execute, request, write_report)=granted)

WRITE REPORT - When a MA writes a new report, the security agent receives a message that includes the requestid and the maid. It then updates the security database. It grants the MA the right to execute the READ REPORT, EDIT REPORT, DELETE REPORT and SEND REPORT tasks for the particular request.

Three parameters define the security context: Access time, the report and the MA. The following security requirements have to be met:

1 The time the MA tries to execute the task has to comply with the rules set for him by his department

2 The MA is granted EDIT REPORT, READ REPORT, SEND REPORT task execution rights

These requirements, which are the Security Agents responsibility, can be expressed as follows:

If ((defined_access_time_from <= User_access_time<=defined_access_time_to) and
(MA, write, report))
then ((MA, execute, report, edit_report)=granted
and (MA, execute, report, read_report)=granted
and (MA, execute, report, send_report)=granted)

SEND REPORT - When a MA decides to send a report, the security agent receives a message that includes the requestid, the maid, the croid of the CRO who has issued the request and the customerid. It then updates the security database. It revokes the MAs access rights to the relevant information of the customer the request is issued for. It also revokes the MAs right to execute EDIT REPORT, DELETE REPORT and SEND REPORT tasks for the particular request. The MA from then on, is only allowed to execute the READ REPORT task for that request. Finally, it grants the CRO the right to execute the READ REPORT and WRITE PROPOSAL tasks for the same request.

Three parameters define the security context: Access time, the report and the MA. The following security requirements have to be met:

1 The time the MA tries to execute the task has to comply with the rules set for him by his department
2 The MAs access customer information rights and task execution rights are revoked, except READ REPORT task execution rights
3 The CRO is granted READ REPORT task execution rights
4 The qualifying MA list is deleted

The first three requirements, which are the Security Agents responsibility, can be expressed as follows:

If ((defined_access_time_from <= User_access_time<=defined_access_time_to)
then ((MA, execute, report, edit_report)=revoked
and (MA, execute, report, send_report)=revoked

and (MA, execute, report, delete_report)=revoked
and (CRO, execute, report, read_report)=granted)

CANCEL REPORT - When a MA decides to cancel a report, the security agent receives a message that includes the requestid, the maid and the croid of the CRO who has issued the request. It then updates the security database. It revokes the MAs rights to execute READ REPORT, EDIT REPORT, DELETE REPORT and SEND REPORT tasks for the particular request. The MA from then on, is only allowed to execute the WRITE REPORT task for that request. Finally, it revokes all the CROs task execution rights for the same request.

Four parameters define the security context: Access time, the report, the CRO and the MA. The following security requirements have to be met:

1 The time the MA tries to execute the task has to comply with the rules set for him by his department
2 The CROs READ REPORT task execution rights are revoked
3 The MAs access customer information rights and task execution rights are granted

These requirement, which is the Security Agents responsibility, can be expressed as follows:

If ((defined_access_time_from <= User_access_time<=defined_access_time_to)
then ((MA, execute, report, write_report)=granted
and (MA, execute, report, read_report)=revoked
and (CRO, execute, report, read_report)=revoked)

WRITE PROPOSAL - When a CRO writes a new proposal, the security agent receives a message that includes the requestid and the croid. It then updates the security database. It grants the CRO the right to execute the READ PROPOSAL, EDIT PROPOSAL, DELETE PROPOSAL and SEND PROPOSAL tasks for the particular request.

Three parameters define the security context: Access time, the proposal and the CRO. The following security requirements have to be met:

1 The time the CRO tries to execute the task has to comply with the rules set for him by his department

2 The CRO is granted EDIT PROPOSAL, READ PROPOSAL, SEND PROPOSAL task execution rights

These requirements, which are the Security Agents responsibility, can be expressed as follows:

If ((defined_access_time_from <= User_access_time<=defined_access_time_to) and
(CRO, write, proposal))
then ((MA, execute, proposal, edit_ proposal)=granted
and (CRO, execute, proposal, read_ proposal)=granted
and (CRO, execute, proposal, send_ proposal)=granted)

SEND PROPOSAL - When a CRO decides to send a proposal, the security agent receives a message that includes the requestid, and the crod. It then updates the security database. It revokes the CROs rights to execute EDIT PROPOSAL, DELETE PROPOSAL and SEND PROPOSAL tasks for the particular request. The CRO from then on is only allowed to execute the READ PROPOSAL task for that request.

Three parameters define the security context: Access time, the proposal and the CRO. The following security requirements have to be met:

1 The time the CRO tries to execute the task has to comply with the rules set for him by his department
2 The CROs task execution rights are revoked, except READ PROPOSAL task execution rights
3 The request state is updated to send.

These requirements, which are the Security Agents responsibility, can be expressed as follows:

If ((defined_access_time_from <= User_access_time<=defined_access_time_to) then ((CRO, execute, proposal, edit_report)=revoked
and (CRO, execute, proposal, delete_report)=revoked
and (CRO, execute, proposal, send_report)=revoked
and (request, send))

CANCEL PROPOSAL - When a CRO decides to cancel a proposal, the security agent receives a message that includes the requestid, the maid, the croid of the CRO who has issued the request and the customerid. It then updates the security database. It revokes all CROs and MAs data and task execution access rights.

Two parameters define the security context: Access time, the request. The following security requirements have to be met:

1 The time the CRO tries to execute the task has to comply with the rules set for him by his department
2 All the respective access and task execution rights have to be removed
3 All the respective assignments have to be removed

The first two requirements, which are the Security Agents responsibility, can be expressed as follows:
If ((defined_access_time_from <= User_access_time<=defined_access_time_to) then ((CRO, execute, request)=revoked
and (MA, execute, request)=revoked
and (MA, customer)=revoked)

The code of the above mentioned JADE AGENT is presented in Apendix B.

7.3 WORKFLOW LAYER

The workflow layer is implemented in JADE platform. A JADE agent accepts messages from the BPEL process, which includes information about the workflow runtime context, and adjust the task assignments accordingly. Whenever a web service that calls for task assignment reconfiguration is executed, the workflow agent takes over, to reflect the new context in the security database.

Its functionality is implemented by five cyclic behaviors and a ticker behavior.

'Cyclic' behaviors are designed to never complete; their action() method executes the same operations each time it is called. Ticker behaviors execute at selected points in time.

The TickerBehaviour has action() and done() methods pre-implemented to execute the onTick() abstract method repetitively, waiting a given period (specified in the constructor) after each execution. A TickerBehaviour never completes unless it is explicitly removed or its stop() method is called.

The workflow agent is used in the following situations:

- Periodically, to check for time critical requests
- When a CRO issues a new request
- When a MA binds a request
- When a MA unbinds a request
- When a MA delegates a request
- When a CRO cancels a request

All other actions do not influence the process context and thus the security requirements and the workflow agent is not activated.

Following, the messages received by the Workflow Agent from the workflow and the Workflow Agent actions are described in more detail:

TIME CRITICAL REQUESTS - Periodically, the workflow agent checks for requests that are unbound after a specific time period has elapsed, since they were issued. These requests are considered as time critical and the workflow agent inserts them to the head of MAs

worklist. The head of the MAs searches for time critical requests and assignees the unbound request to one of the originally qualifying MAs.

One parameter defines the security context: Time. So, one security requirement has to be met:

1 All delayed requests have to be identified and assigned to the head of MAs

This requirement, which is the Workflow Agents responsibility, can be expressed as follows:

If (request, date - date_request_send >5) then request in urgent_request_set

SEND REQUEST - When a new request is issued, the workflow agent receives a message that includes the requestid , the croid and the customerid. It consults the respective business rules and assigns the request to the MAs that rules evaluate to OK.

When a CRO sends a request, a new process instance is created. The Security and Workflow Agents have to resolve the following security issues:

a. Which MAs will be allowed to handle the request?
b. Which is the subset of the above MAs that the business rules will allow the request to be forwarded to?

The following parameters define the security context: Access time, ownership, the speciality of the request and the predefined business rules which are evaluated against the following parameters: Current MA workload and current MA availability. So, the following requirements have to be met:

1 The MAs that have the request speciality are allowed to handle the request
2 The MAs that will be available for a certain number of days in the time period in which the request must be answered (e.g. not unavailable for more than five days for that time period), are initially qualified
3 The less burdened MAs are finally assigned with the request.

The last two requirements, which is the Workflow Agents responsibility, can be expressed as follows:

If ((MA in qualifying_MA_set) and (MA_availability>=(date_request_send – date_request_answered-5))) then MA in qualifying_MA_subset

If ((MA in qualifying_MA_subset) and (MA_workload<=average(MA_set_workload))) then MA in final_qualifying_MA_subset

BIND REQUEST - When a MA binds a request, the workflow agent receives a message that includes the requestid, and the maid. It then updates the security database. It changes the state of the request, it removes it from the worklist of all the originally qualifying MAs except the one who bound it and updates the request record with the maid.

Three parameters define the security context: Access time, the request and the MA. So, three security requirements have to be met:

1 Only a specific MA can access the request
2 The time the MA tries to execute the task has to comply with the rules set for him by his department
3 That MA has access rights to the respective customer information and execution rights to WRITE REPORT task.

The first requirement, which is the Workflow Agents responsibility, can be expressed as follows:

If (MA, bind , request) then access=granted else access =revoked

UNBIND REQUEST - When a MA unbinds a request, the workflow agent receives a message that includes the requestid. It then updates the security database. It changes the state of the request, deletes maid from the request record and reassigns it to all the originally qualifying the MAs.

Three parameters define the security context: Access time, the request and the MA. three security requirements have to be met:

1 All the initially qualifying MAs can access the request
2 The time the MA tries to execute the task has to comply with the rules set for him by his department
3 Not MA has access rights to the respective customer information and execution rights to WRITE REPORT task.

The first requirement, which is the Workflow Agents responsibility, can be expressed as follows:

If (MA, assign , request) then access=granted

DELEGATE REQUEST - When a MA delegates a request, the workflow agent receives a message that includes the requestid, the maid, and a second maid of the MA the request will be delegated to. It then updates the security database. It updates the maid in the request record, removes the assignment from the first MA and assigns the request to the second MA.

Four parameters define the security context: Access time, the request, the delegator and the degatee. Three security requirements have to be met:

1 Only the MA that will be delegated with the request will be able to access it
2 The time the MA tries to execute the task has to comply with the rules set for him by his department
3 That MA has access rights to the respective customer information and execution rights to WRITE REPORT task.

The first requirement, which is the Workflow Agents responsibility, can be expressed as follows:

If (MA, delegate , request) then access=granted else access =revoked

CANCEL REQUEST - When a CRO cancels a request, the workflow agent receives a message that includes the requestid. It then changes the state of the request, removes all assignments from the CRO and the MAs relating to that request and deletes all relevant records.

Two parameters define the security context: Access time, the request. The following security requirements have to be met:

1 The time the CRO tries to execute the task has to comply with the rules set for him by his department
2 All the respective access and task execution rights have to be removed
3 All the respective assignments have to be removed

The third requirement, which is the Workflow Agents responsibility, can be expressed as follows:
If (MA, assign, request) then assign=revoked

The code of the above mentioned JADE AGENT is presented in Apendix C.

7.4 BUSINESS PROCESS LAYER

The process layer is implemented in BPEL (Business Process Execution Language for Web Services). BPEL is a specialized language for composing services into business processes that provides the ability to express business processes in a standardized way. BPEL stimulates enterprises to further define their business processes, which in turn leads to business process optimization, reengineering, and the selection of the most appropriate processes, thus further optimizing the organization.

The business process or the presented case study, can be separated in three main subprocesses

- Issue Request
- Write Report
- Write Proposal.

These subprocesses can be further analyzed in the individual tasks that need to be executed for them to be completed.

Each subprocess may listen for events, the occurrence of which diverts the regular flow of the process and causes the execution of predefined operations in order to cope with that event. Some of these events are:

- Cancel Request
- Cancel Report
- Unbind Request
- Assign Request

The process contains the following scopes, which are a lower level of analysis.

- Issue_Request
- Bind_Assign_Request
- Report
- Write_Proposal
- Send_Proposal

Figure 8 presents the overall business process.

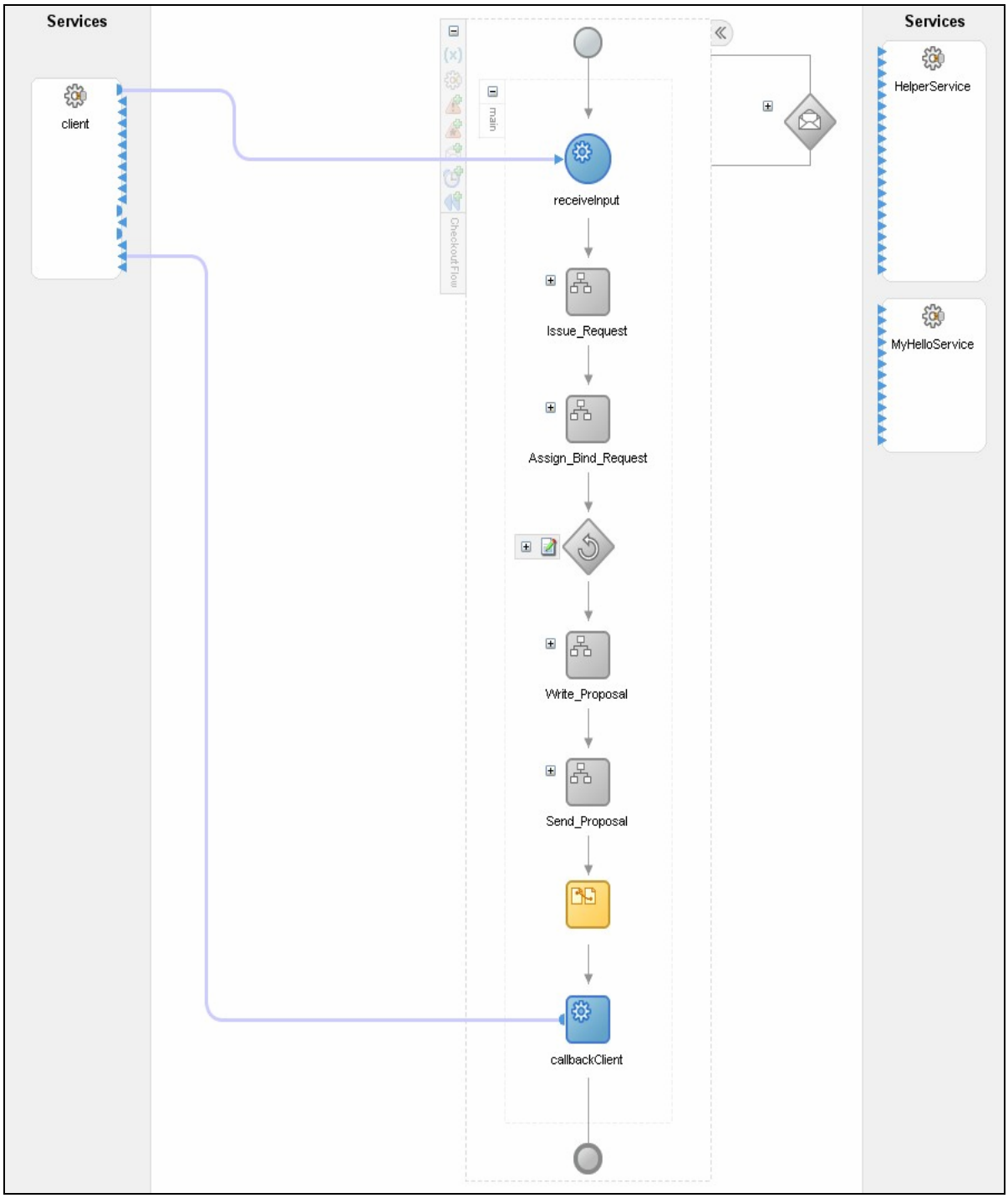


Figure 8. High Level Business Process

When a CRO sends a new request, a new process instance is created and the process input variable is initialized. Through the flow of the process, the input variable is updated, in order to reflect the parameters of the particular instance.

```
<complexType name="orderType">
        <sequence>
<element name="RequestId" type="string"/>
<element name="ConversationId" type="string"/>
<element name="MaId" type="string"/>
<element name="CustomerId" type="string"/>
<element name="MaDelegId" type="string"/>
<element name="Spec" type="string"/>
<element name="CroId" type="string"/>
<element name="InvokeWsif" type="string"/>
<element name="Wf_S_Agent" type="string"/>
<element name="ProductId" type="string"/>
<element name="Quantity" type="string"/>
<element name="Memo" type="string"/>
<element name="EmployeeId" type="string"/>
<element name="InvokeWeb" type="string"/>
        </sequence>
</complexType>
```

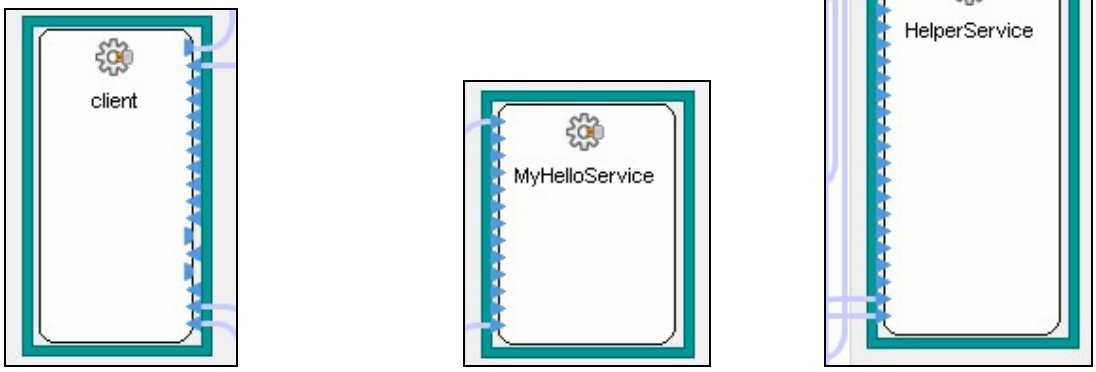
The BPEL process interacts with three services participating in it.

- The client
- The Web Service
- The JADE Agents

These interactions occur through three Partner Links, which connect the process with the outside world.

```
<partnerLinks>
  <!--
      The 'client' role represents the requester of this service. It is
      used for callback. The location and correlation information associated
      with the client role are automatically set using WS-Addressing.
      -->
  <partnerLink name="client" partnerLinkType="tns:CheckoutFlow"
            myRole="CheckoutFlowProvider"
            partnerRole="CheckoutFlowRequester"/>
  <partnerLink name="HelperService" partnerRole="HelperServiceProvider"
            partnerLinkType="services:HelperService"/>
  <partnerLink name="MyHelloService" partnerRole="IssueProvider"
            partnerLinkType="ns1:IssueService"/>
 </partnerLinks>
```

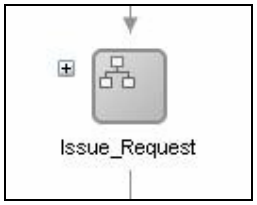
The Partner Links are presented in Figure 9:



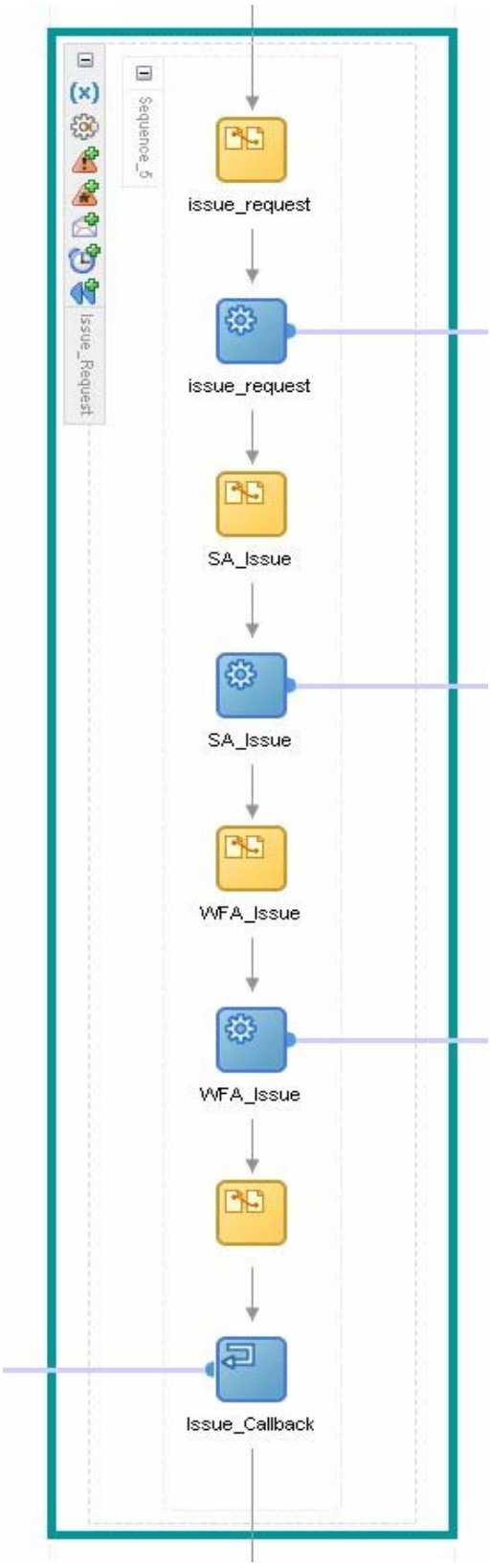
Client Partner Link Web Service Partner Link JADE Agent Partner Link

Figure 9 Partner Links

The first scope of the business process is the Issue_Request scope.



Following, the IssueRequest scope is analyzed in a lower level.



The input parameters of the issuerequest web service are assigned values from the process input variable.

Next, the issuerequest web service is invoked through the respective Partner Link.

The input parameters of the Security Agent are assigned values from the process input variable, the issuerequest web service output variable and a predefined value that defines the Agent behavior that will be executed.

Next, the Security Agent is invoked through the respective Partner Link.

The input parameters of the Workflow Agent are assigned values from the process input variable, the issuerequest web service output variable and a predefined value that defines the Agent behavior that will be executed.

Next, the workflow Agent is invoked through the respective Partner Link.

The return parameter is assigned a value from the process input parameter.

A reply is returned to the Client through the respective Partner Link.

Figure 10. Issue_Request Scope

Following, each activity is analyzed in more detail.

For every <receive> activity, we have to define:

- The activity name
- The Partner Link
- The BPEL operation the client wishes to invoke
- The input variable
- Whether a new process instance will be created (it is checked only for the initial operation)

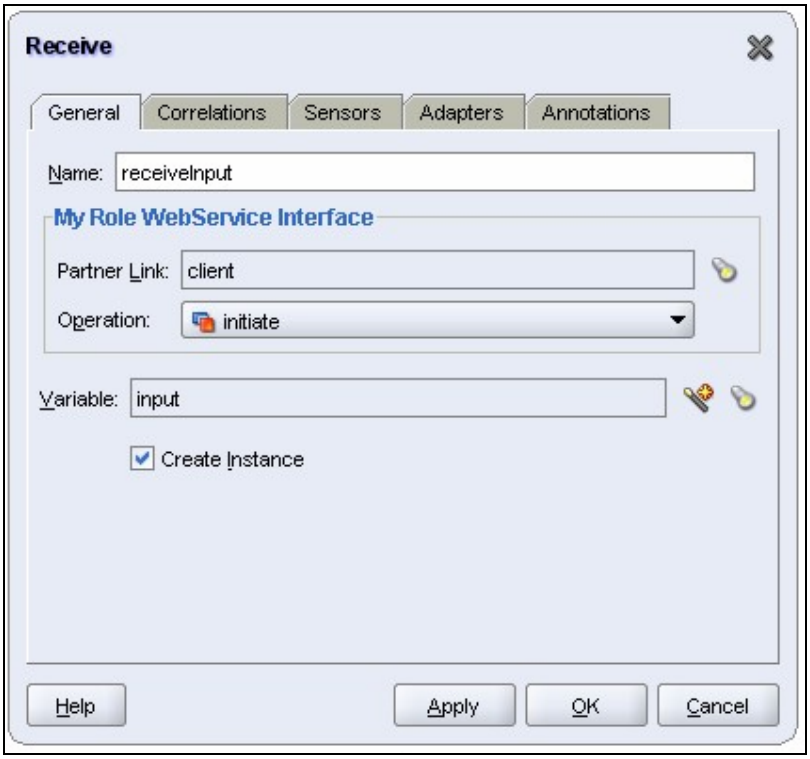


Figure 11. <receive> Activity Parameters

After every <receive> activity, the input variable that was received may be assigned to other variables (in this case, the process input variable) through an <assign> activity.

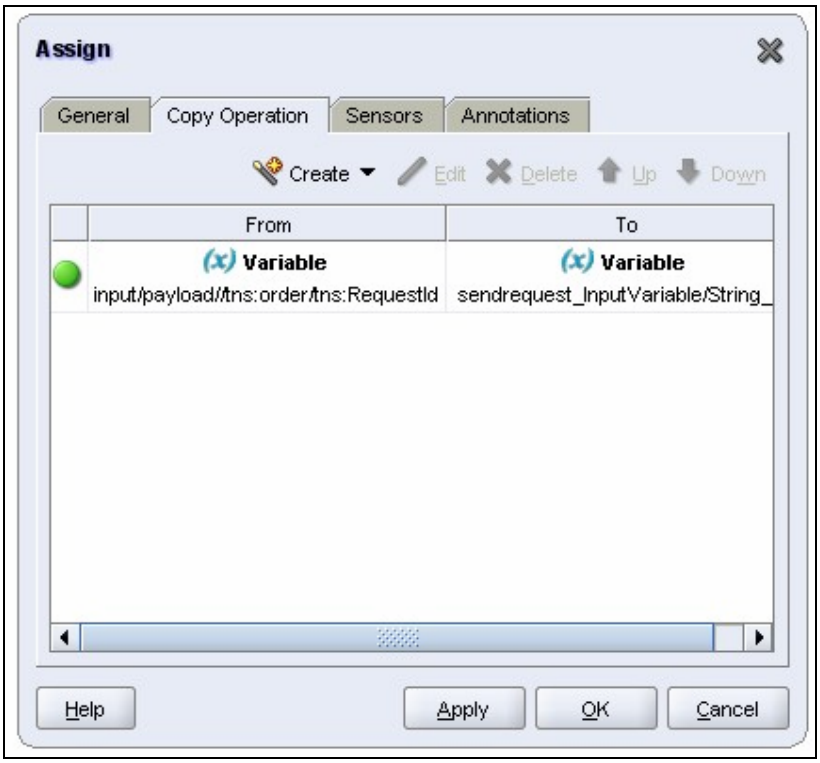


Figure 12. <assign> Activity

Before every <invoke> activity, we have to assign values to the input parameters of the operation that will be invoked. This is accomplished with the help of an <assign> activity.

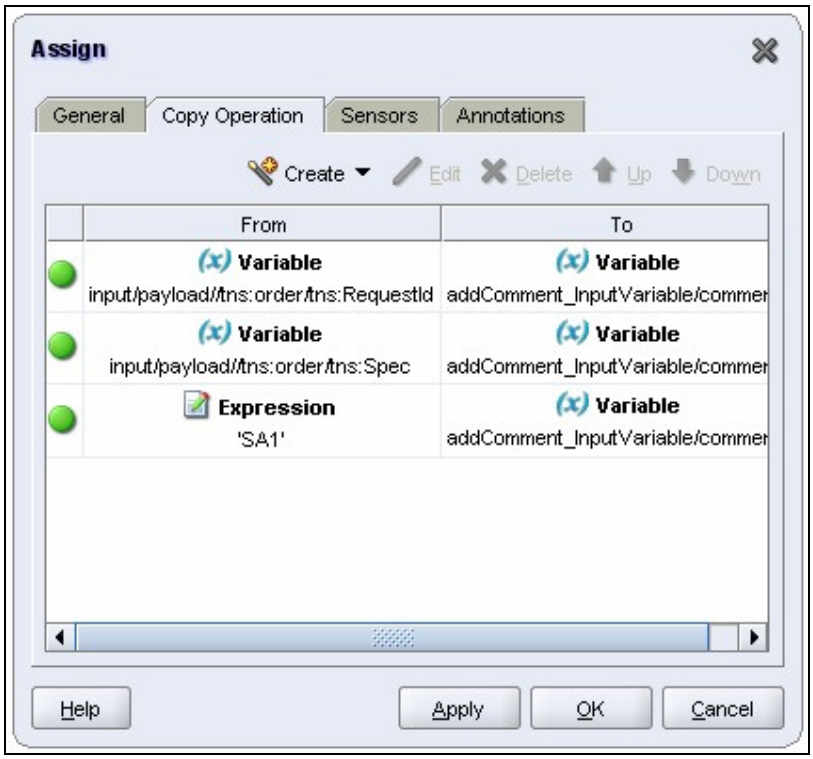


Figure 13. <assign> Activity

For every <invoke> activity, we have to define:

- The activity name
- The Partner Link
- The operation the BPEL process wishes to invoked
- The input variable
- The output variable

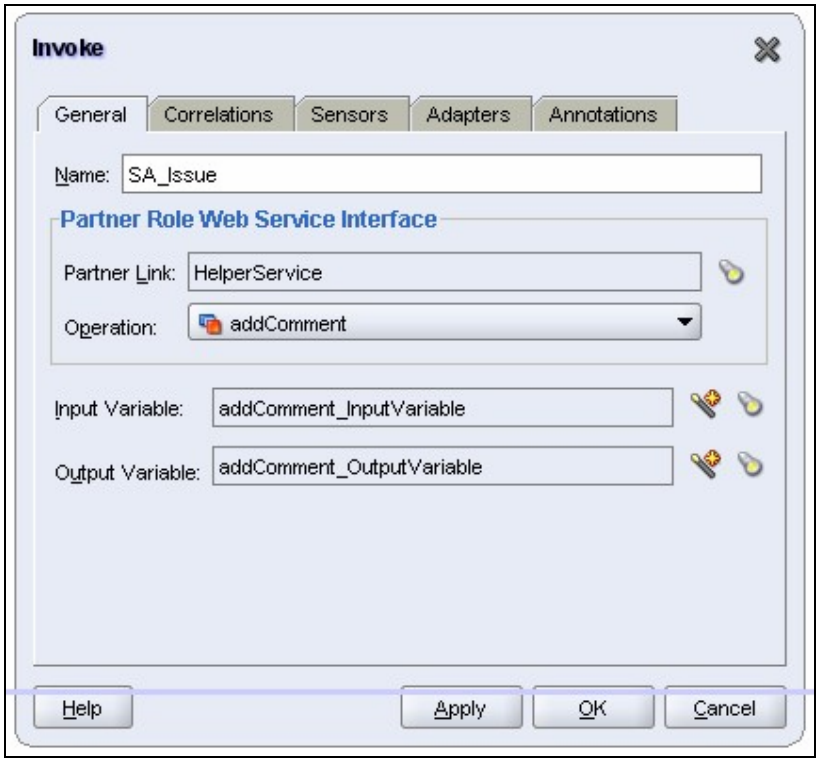


Figure 14. <invoke> Activity Parameters

After every <invoke> activity, the output variable of the operation that was invoked may have to be assigned to another process variable which is accomplished through an <assign> activity.

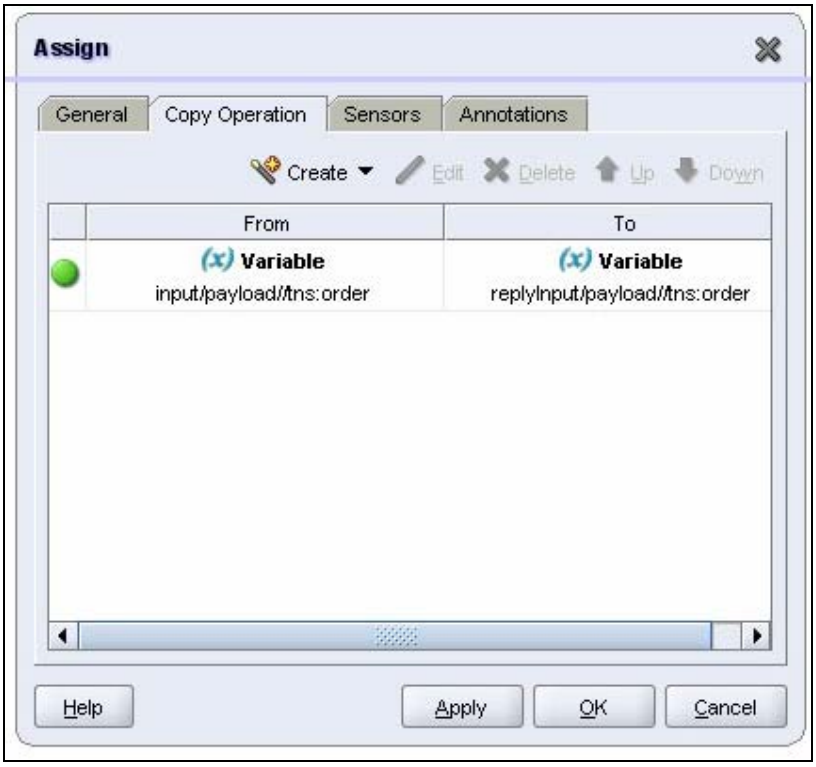


Figure 15. <assign> Activity

Every time a response has to be returned to a Client, the output parameter has to be assigned an appropriate value with the help of an <assign> activity.

Next, a <reply> activity has to be used for the callback operation. For every <reply> activity, we have to define:

- The activity name
- The Partner Link
- The operation the callback corresponds to
- The return variable

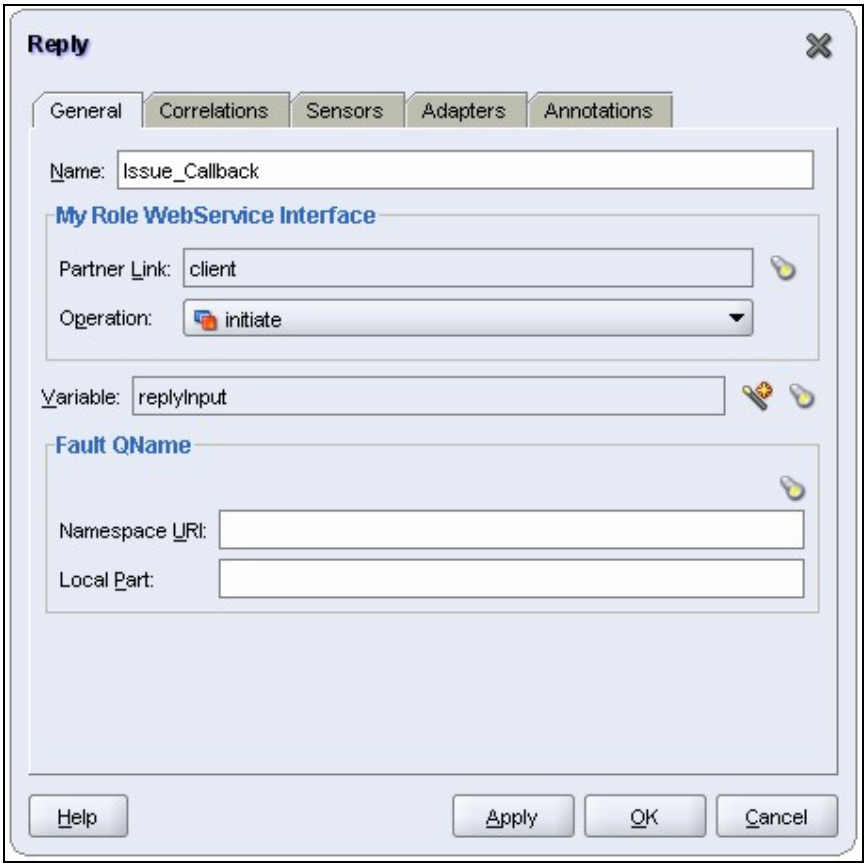


Figure 16. <reply> Activity Parameters

After a new request has been issued and the Issue_Request scope has been completed, the business process blocks in the next scope (Bind_Assign_Request Scope) and waits for a client to invoke one of two operations

- Bind Request
- Assign Request

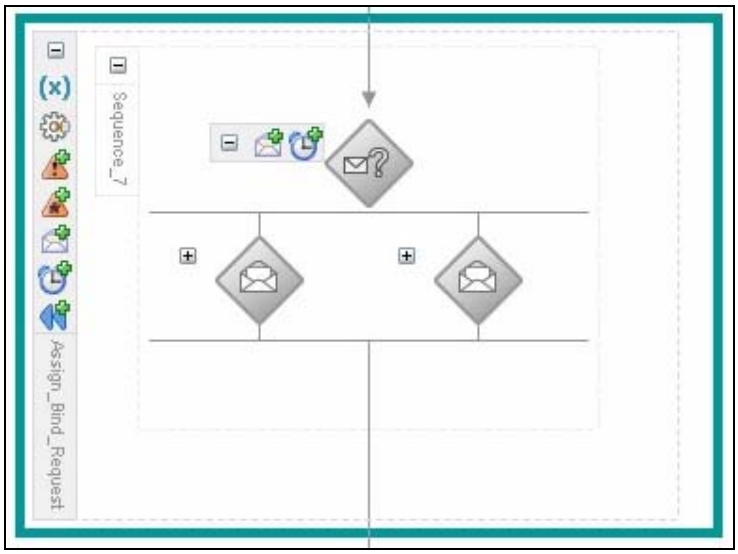
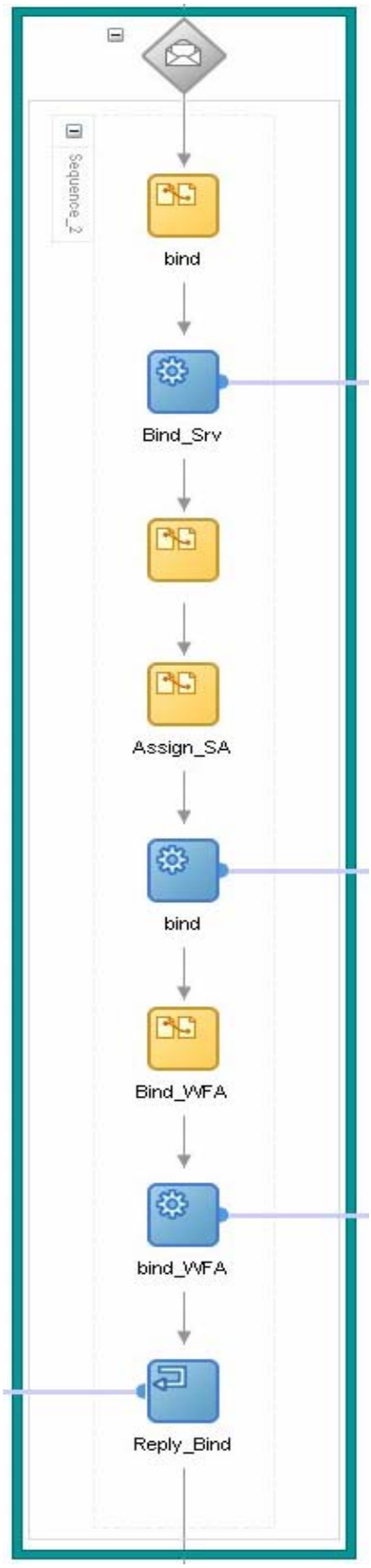


Figure 17. Bind_Assign_Request Scope

The Security and Workflow agents have updated the worklists of the qualifying MAs with the new request Issue_Request scope. So, when a MA searches for new requests that are assigned to him, he can see the new entry.

If a MA decides to answer to a request, he binds it. This action causes the initialization of the input parameter of the BPEL process bind operation and the invocation of that operation. The process unblocks and enters the Bind_Request scope.

Following, the Bind_Request scope is analyzed in a lower level.



A message from the Client Partner Link is received, that invokes the bind BPEL process operation

The input parameters of the bindrequest web service are assigned values from the client input variable.

Next, the bindrequest web service is invoked through the respective Partner Link.

The bindrequest service output variable is assigned to the process input variable.

The input parameters of the Security Agent are assigned values from the process input variable, the bindrequest web service output variable and a predefined value that defines the Agent behavior that will be executed.

Next, the Security Agent is invoked through the respective Partner Link.

The input parameters of the Workflow Agent are assigned values from the process input variable, the bindrequest web service output variable and a predefined value that defines the Agent behavior that will be executed.

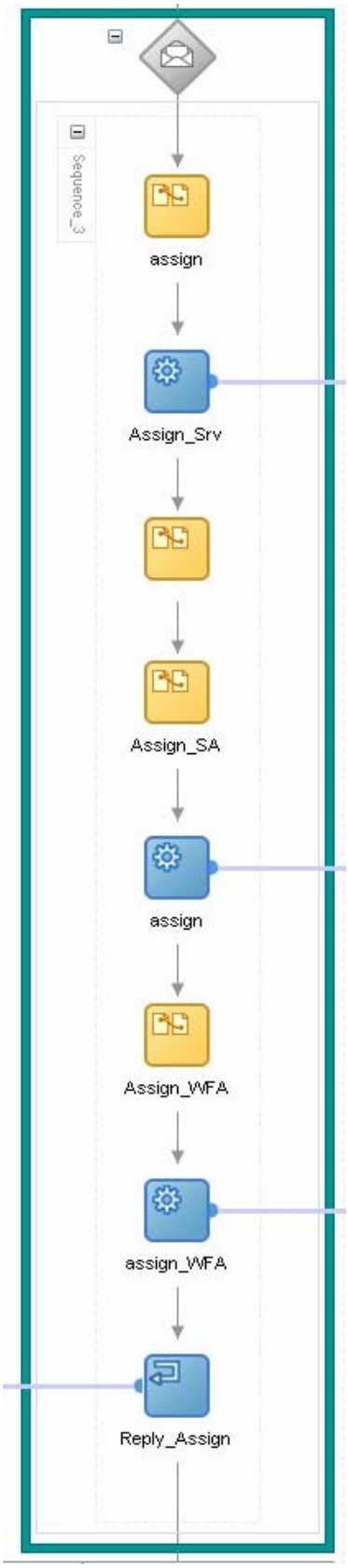
Next, the workflow Agent is invoked through the respective Partner Link.

A reply is returned to the Client through the respective Partner Link.

Figure 18. Bind_Request Scope

If after a specified period of time has elapsed since a request was issued and the request is unbound, it is placed by the Workflow agent in the worklist of the head of MAs. The head of MAs searches for time critical requests and assigns them to MAs who where initially considered qualified to answer them. This action causes the initialization of the input parameter of the BPEL process assign operation and the invocation of that operation. The process unblocks and enters the Assign_Request scope.

Following, the Assign _Request scope is analyzed in a lower level.



A message from the Client Partner Link is received, that invokes the assign BPEL process operation

The input parameters of the assignrequest web service are assigned values from the client input variable.

Next, the assignrequest web service is invoked through the respective Partner Link.

The assignrequest service output variable is assigned to the process input variable.

The input parameters of the Security Agent are assigned values from the process input variable, the assignrequest web service output variable and a predefined value that defines the Agent behavior that will be executed.

Next, the Security Agent is invoked through the respective Partner Link.

The input parameters of the Workflow Agent are assigned values from the process input variable, the assignrequest web service output variable and a predefined value that defines the Agent behavior that will be executed.

Next, the workflow Agent is invoked through the respective Partner Link.

A reply is returned to the Client through the respective Partner Link.

Figure 19. Assign_Request Scope

After a request has been bound or assigned, and the Bind_Assign_Request scope has been completed, the business process blocks in the next scope (Report Scope) and waits for a client to invoke the BPEL process operation write_report.

At the same time, the process listens for a list of events (client invocations):

- The MA who has bound the request unbinds it (Unbind BPEL process operation)
- Another MA binds the previously unbound request (Bind BPEL process operation)
- The head of MAs assigns the previously unbound request (Assign BPEL process operation)
- The MA who has bound the request delegates it (delegate BPEL process operation)
- The MA who has written the report, deletes it (Delete_Report BPEL process operation)
- The MA who has written the report, sends it to the CRO who issued the request (Send_Report BPEL process operation)

Figure 20 presents the report scope.

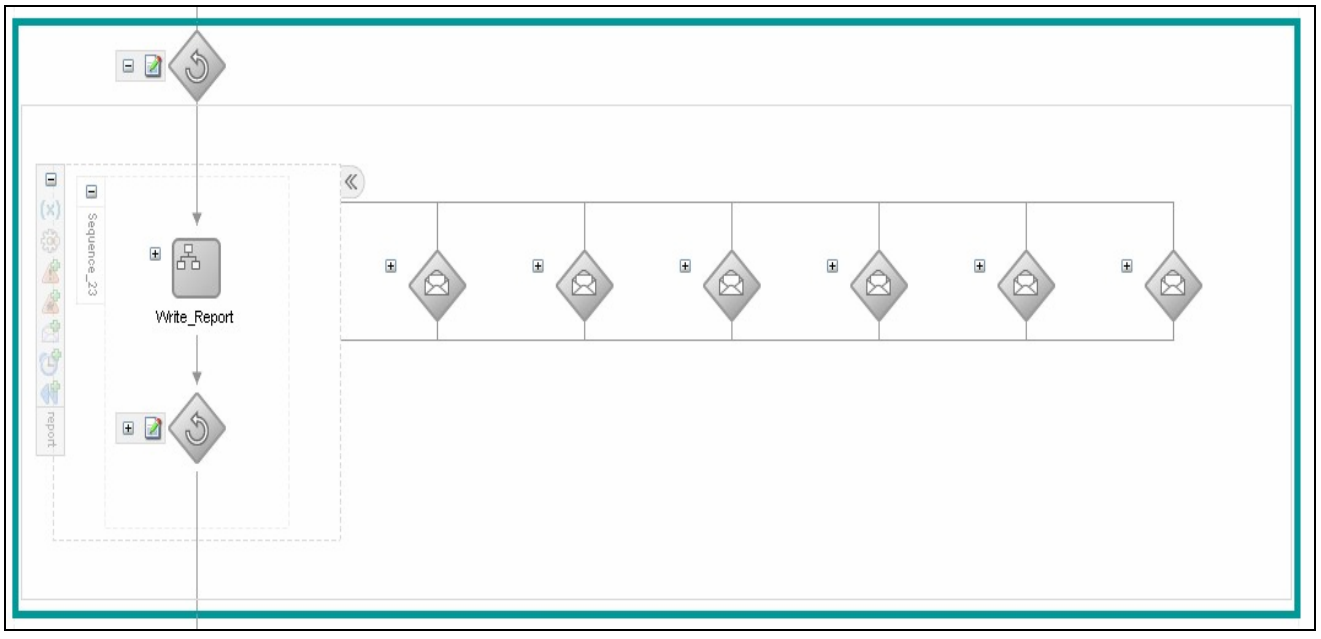
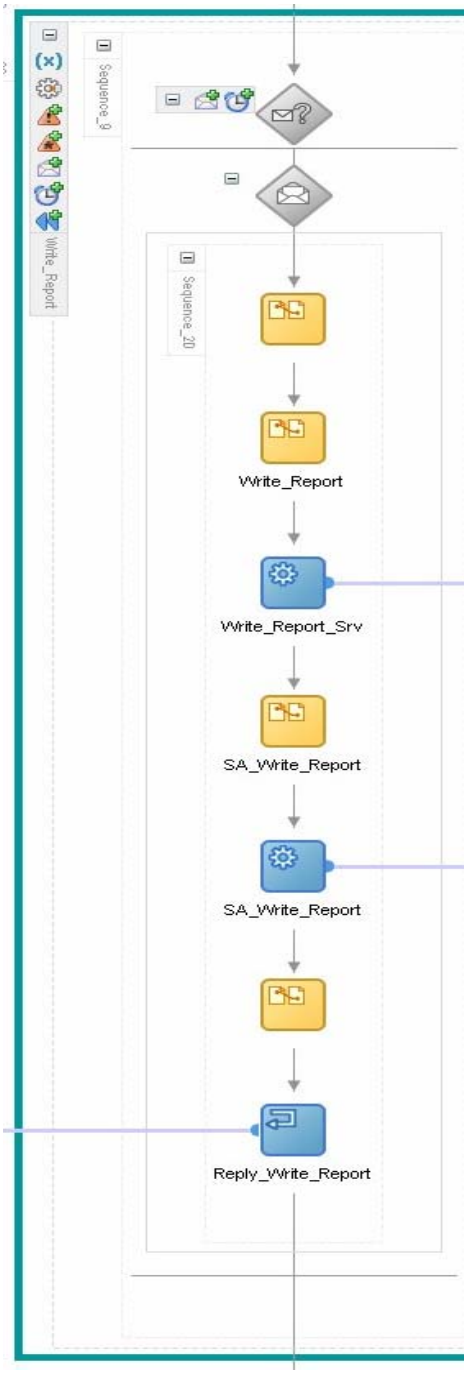


Figure 20. Report Scope

The MA who bound the request decides to white the corresponding report. This action causes the initialization of the input parameter of the BPEL process write-report operation and the invocation of that operation. The process unblocks and enters the Report scope.

Following, the Report scope is analyzed in a lower level. First, the write_report subscope is presented.



A message from the Client Partner Link is received, that invokes the write_report BPEL process operation

The client input variable is assigned to the process input variable

The input parameters of the addreport web service are assigned values from the process input variable.

Next, the addreport web service is invoked through the respective Partner Link.

The input parameters of the Security Agent are assigned values from the process input variable, the addreport web service output variable and a predefined value that defines the Agent behavior that will be executed.

Next, the Security Agent is invoked through the respective Partner Link.

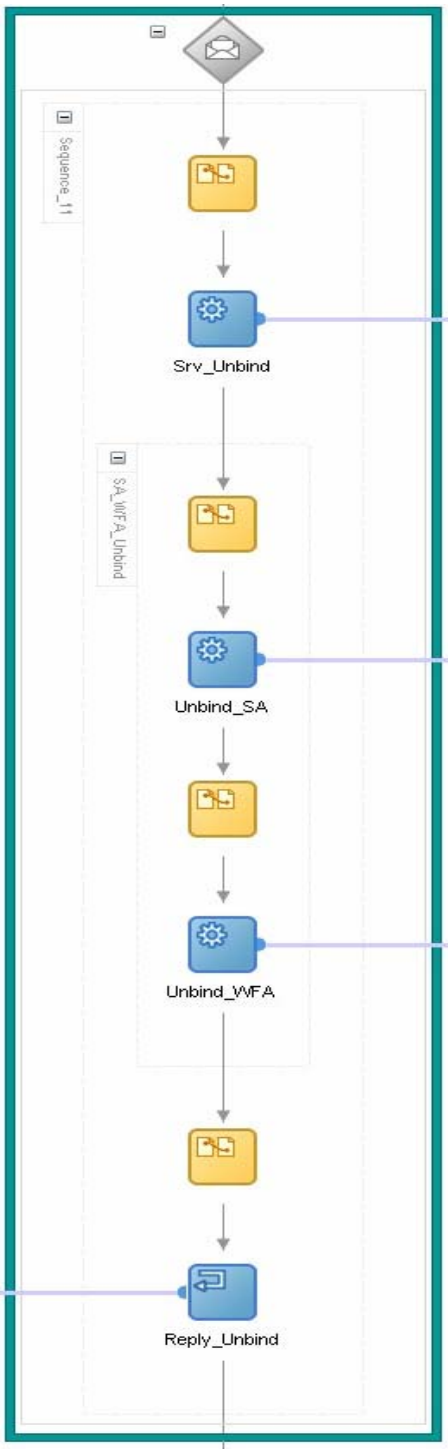
The output variable of the operation is initialized.

A reply is returned to the Client through the respective Partner Link.

Figure 21. Write_Report Scope

The MA who bound the request decides to unbind it. This action causes the initialization of the input parameter of the BPEL process unbind operation and the invocation of that operation. The process that is blocked in the report scope and listens for events, unblocks and enters the unbind_request scope.

Following, the unbind_request scope is analyzed in a lower level.



A message from the Client Partner Link is received, that invokes the unbind BPEL process operation

The input parameters of the unbind_request web service are assigned values from the client input variable.

Next, the unbind_request web service is invoked through the respective Partner Link.

The input parameters of the Security Agent are assigned values from the process input variable, the unbind_request web service output variable and a predefined value that defines the Agent behavior that will be executed.

Next, the Security Agent is invoked through the respective Partner Link.

The input parameters of the Workflow Agent are assigned values from the process input variable, the unbind_request web service output variable and a predefined value that defines the Agent behavior that will be executed.

Next, the workflow Agent is invoked through the respective Partner Link.

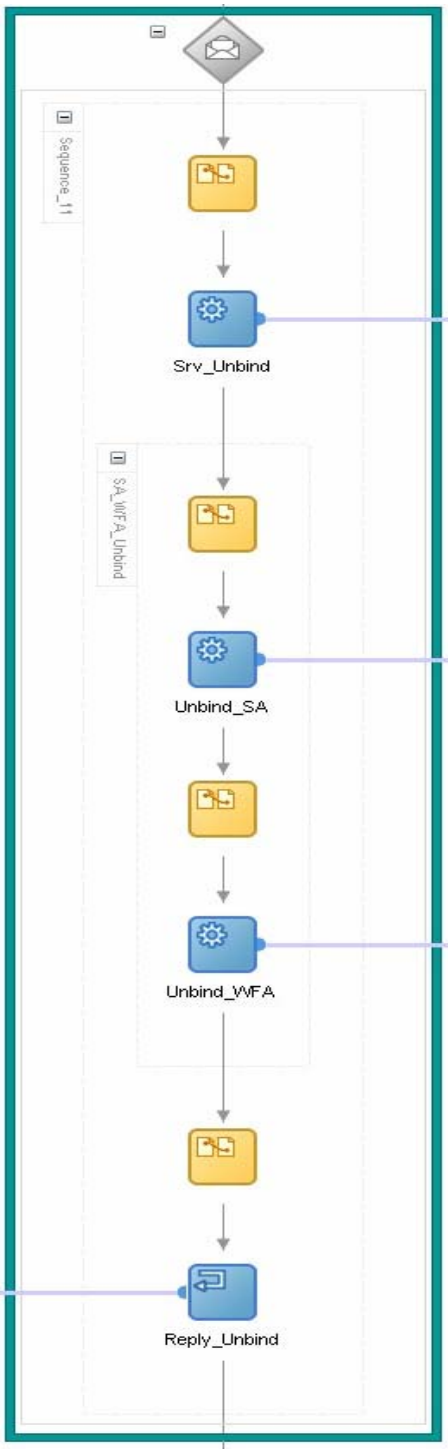
The output parameter of the unbind BPEL process operation is initialized

A reply is returned to the Client through the respective Partner Link.

Figure 22. Unbind_Request Scope

A MA decides to bind the previously unbound request. This action causes the initialization of the input parameter of the BPEL process bind operation and the invocation of that operation. The process that is blocked in the report scope and listens for events, unblocks and enters the bind_request scope.

Following, the bind_request scope is analyzed in a lower level.



A message from the Client Partner Link is received, that invokes the bind BPEL process operation

The input parameters of the bind_request web service are assigned values from the client input variable.

Next, the bind_request web service is invoked through the respective Partner Link.

The input parameters of the Security Agent are assigned values from the process input variable, the bind_request web service output variable and a predefined value that defines the Agent behavior that will be executed.

Next, the Security Agent is invoked through the respective Partner Link.

The input parameters of the Workflow Agent are assigned values from the process input variable, the bind_request web service output variable and a predefined value that defines the Agent behavior that will be executed.

Next, the workflow Agent is invoked through the respective Partner Link.

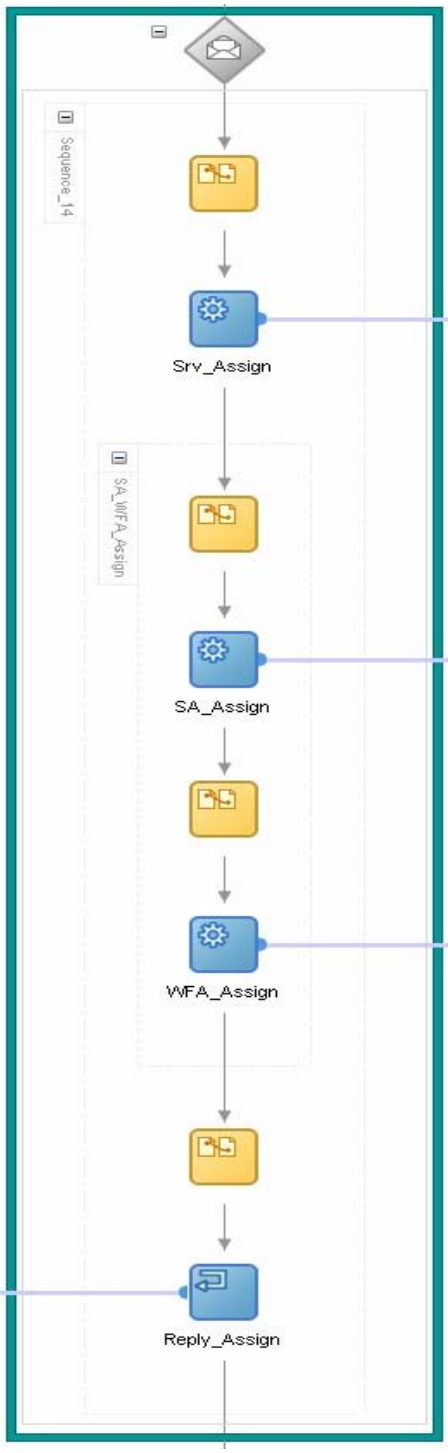
The output parameter of the bind BPEL process operation is initialized

A reply is returned to the Client through the respective Partner Link.

Figure 23. Bind_Request Scope

A head of MAs assigns the previously unbound request. This action causes the initialization of the input parameter of the BPEL process assign operation and the invocation of that operation. The process, which is blocked in the report scope and listens for events, unblocks and enters the assign_request scope.

Following, the assign_request scope is analyzed in a lower level.



A message from the Client Partner Link is received, that invokes the assign BPEL process operation

The input parameters of the assign_request web service are assigned values from the client input variable.

Next, the assign_request web service is invoked through the respective Partner Link.

The input parameters of the Security Agent are assigned values from the process input variable, the assign_request web service output variable and a predefined value that defines the Agent behavior that will be executed.

Next, the Security Agent is invoked through the respective Partner Link.

The input parameters of the Workflow Agent are assigned values from the process input variable, the assign_request web service output variable and a predefined value that defines the Agent behavior that will be executed.

Next, the workflow Agent is invoked through the respective Partner Link.

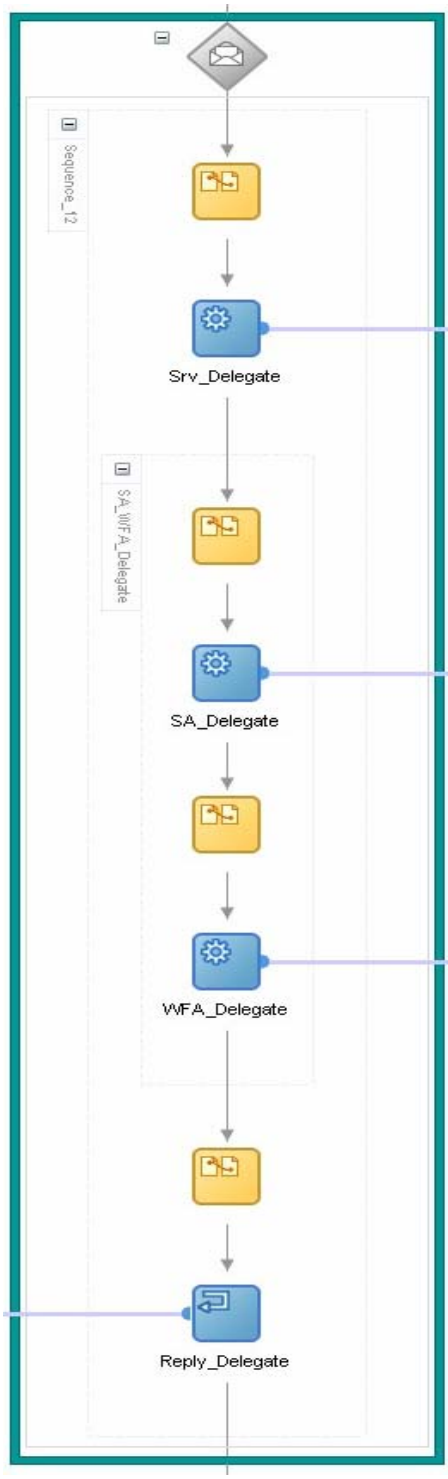
The output parameter of the assign BPEL process operation is initialized

A reply is returned to the Client through the respective Partner Link.

Figure 24. Assign_Request Scope

A MA decides to delegate a request he has bounded. This action causes the initialization of the input parameter of the BPEL process delegate operation and the invocation of that operation. The process that is blocked in the report scope and listens for events, unblocks and enters the delegate_request scope.

Following, the delegate_request scope is analyzed in a lower level.



A message from the Client Partner Link is received, that invokes the delegate BPEL process operation

The input parameters of the delegate_request web service are assigned values from the client input variable.

Next, the delegate_request web service is invoked through the respective Partner Link.

The input parameters of the Security Agent are assigned values from the process input variable, the delegate_request web service output variable and a predefined value that defines the Agent behavior that will be executed.

Next, the Security Agent is invoked through the respective Partner Link.

The input parameters of the Workflow Agent are assigned values from the process input variable, the delegate_request web service output variable and a predefined value that defines the Agent behavior that will be executed.

Next, the workflow Agent is invoked through the respective Partner Link.

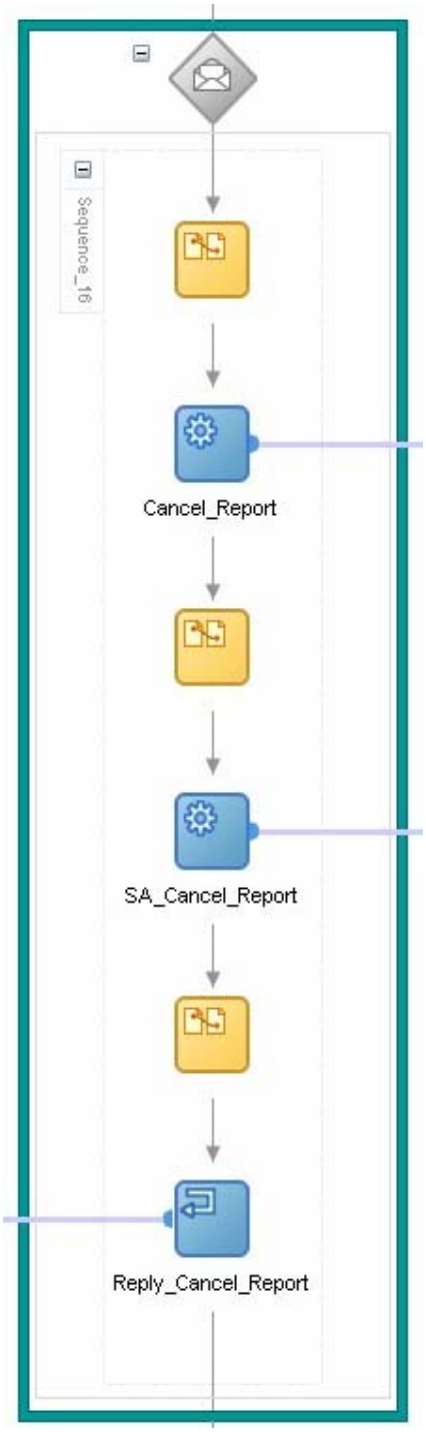
The output parameter of the delegate BPEL process operation is initialized

A reply is returned to the Client through the respective Partner Link.

Figure 25. Delegate_Request Scope

The MA who has written a report for the bound request, decides to delete it. This action causes the initialization of the input parameter of the BPEL process delete_report operation and the invocation of that operation. The process that is blocked in the report scope and listens for events, unblocks and enters the delete_report scope.

Following, the delete_report scope is analyzed in a lower level.



A message from the Client Partner Link is received, that invokes the delete_report BPEL process operation

The input parameters of the delete_report web service are assigned values from the client input variable.

Next, the delete_report web service is invoked through the respective Partner Link.

The input parameters of the Security Agent are assigned values from the process input variable, the delete_report web service output variable and a predefined value that defines the Agent behavior that will be executed.

Next, the Security Agent is invoked through the respective Partner Link.

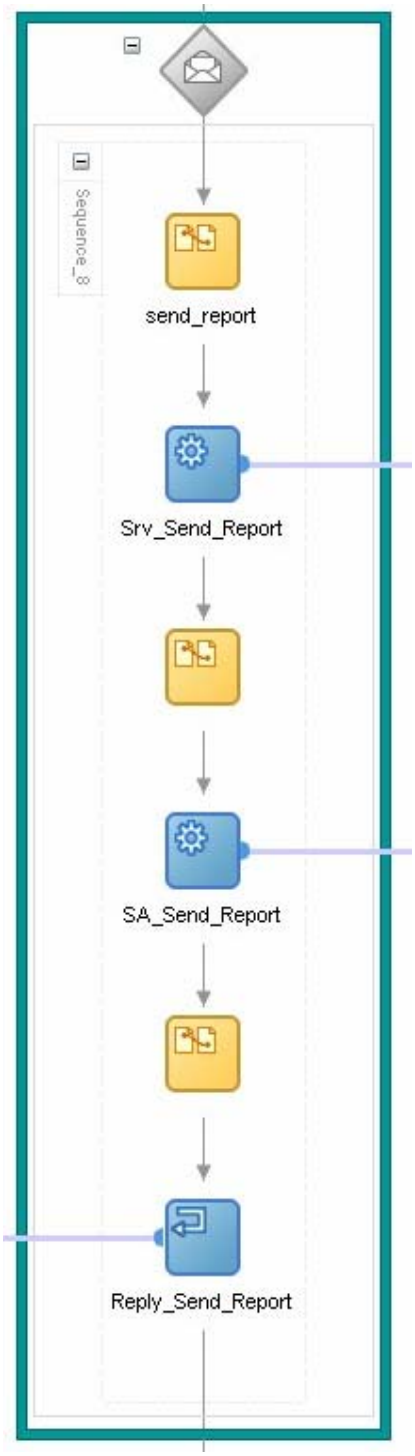
The output parameter of the delete_report BPEL process operation is initialized

A reply is returned to the Client through the respective Partner Link.

Figure 26. Delete_Report Scope

A MA decides to send a report he has written. This action causes the initialization of the input parameter of the BPEL process send_report operation and the invocation of that operation. The process that is blocked in the report scope and listens for events, unblocks and enters the send_report scope.

Following, the send_report scope is analyzed in a lower level.



A message from the Client Partner Link is received, that invokes the delegate BPEL process operation

The input parameters of the send_report web service are assigned values from the client input variable.

Next, the send_report web service is invoked through the respective Partner Link.

The input parameters of the Security Agent are assigned values from the process input variable, the send_report web service output variable and a predefined value that defines the Agent behavior that will be executed.

Next, the Security Agent is invoked through the respective Partner Link.

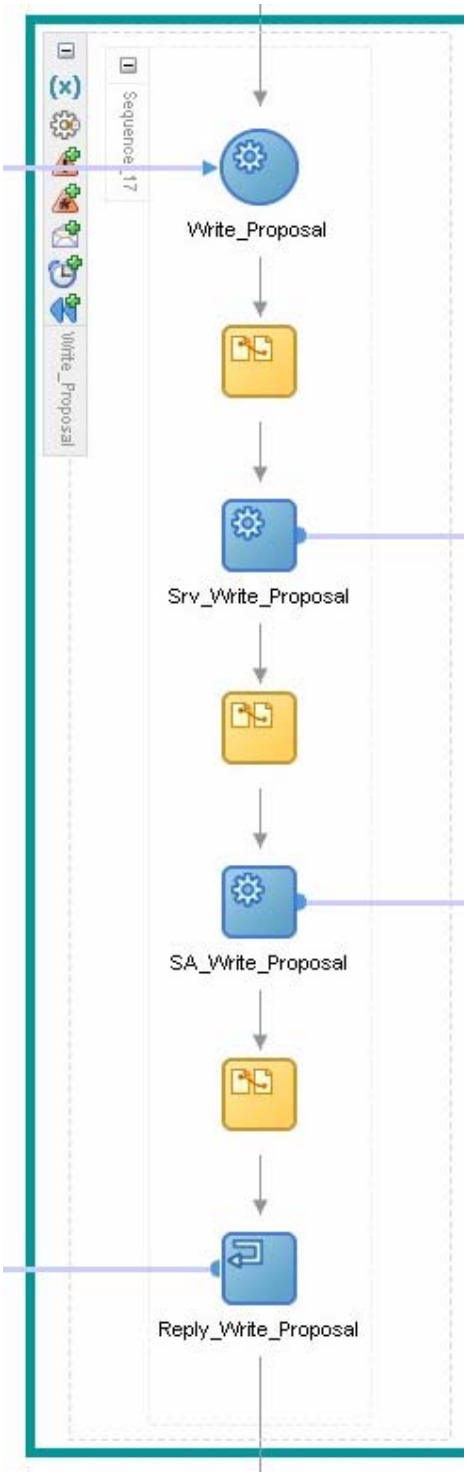
The output parameter of the send_report BPEL process operation is initialized

A reply is returned to the Client through the respective Partner Link.

Figure 27. Send_Report Scope

When the MA sends the report, the Report scope is exited and the BPEL process blocks in the next scope (Write_proposal), waiting for the respective CRO to write the proposal.

When a CRO decides to write the proposal, this action, causes the initialization of the input parameter of the BPEL process write_proposal operation and the invocation of that operation. Following, the write_proposal scope is analyzed in a lower level.



A message from the Client Partner Link is received, that invokes the write_proposal BPEL process operation

The input parameters of the write_proposal web service are assigned values from the client input variable.

Next, the write_proposal web service is invoked through the respective Partner Link.

The input parameters of the Security Agent are assigned values from the process input variable, the write_proposal web service output variable and a predefined value that defines the Agent behavior that will be executed.

Next, the Security Agent is invoked through the respective Partner Link.

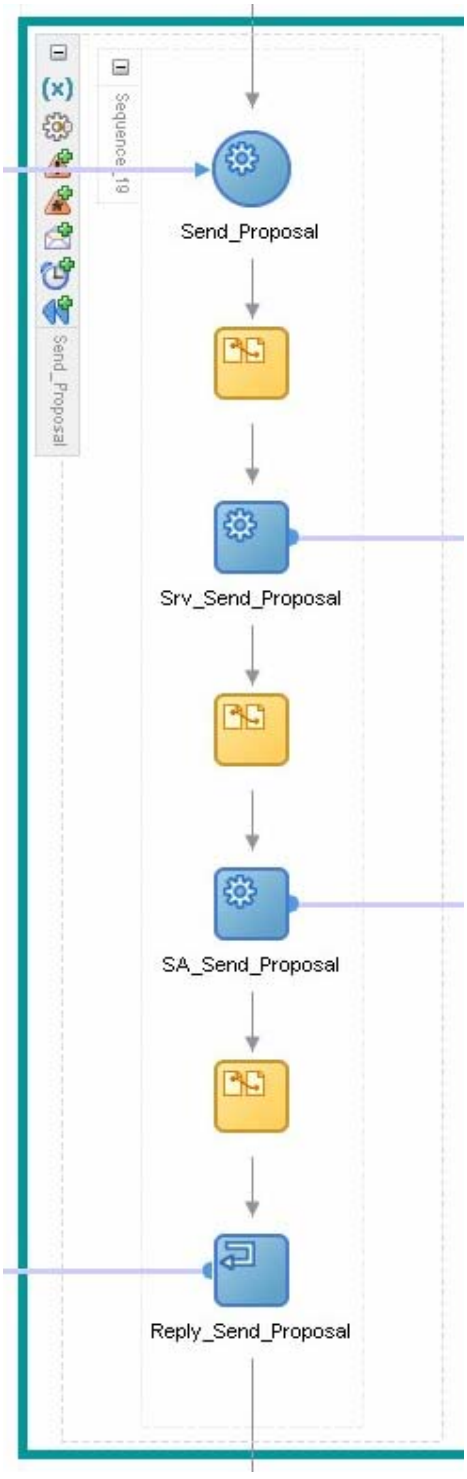
The output parameter of the write_proposal BPEL process operation is initialized

A reply is returned to the Client through the respective Partner Link.

Figure 28. Write_Proposal Scope

After the proposal has been written, the process blocks in the next scope (send_proposal), waiting for the respective CRO to send the proposal.

When a CRO decides to send the proposal, this action, causes the initialization of the input parameter of the BPEL process send_proposal operation and the invocation of that operation. Following, the send_proposal scope is analyzed in a lower level.



A message from the Client Partner Link is received, that invokes the send _proposal BPEL process operation

The input parameters of the send _proposal web service are assigned values from the client input variable.

Next, the send _proposal web service is invoked through the respective Partner Link.

The input parameters of the Security Agent are assigned values from the process input variable, the send _proposal web service output variable and a predefined value that defines the Agent behavior that will be executed.

Next, the Security Agent is invoked through the respective Partner Link.

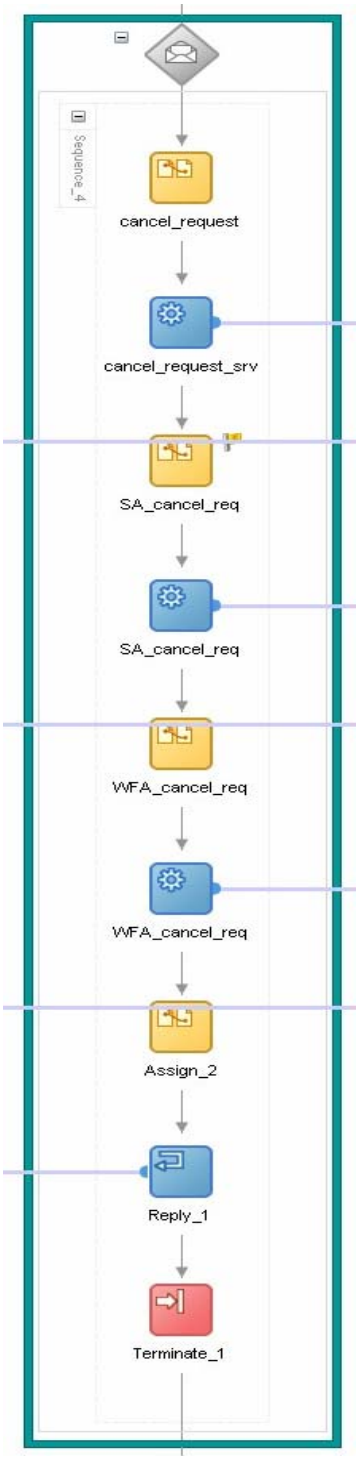
The output parameter of the send _proposal BPEL process operation is initialized

A reply is returned to the Client through the respective Partner Link.

Figure 29. Send_Proposal Scope

From the moment the process instance is created, the BPEL process is listening for the event cancel_request, which occurs when the CRO who issued the request, decides to cancel it. This action causes the initialization of the input parameter of the BPEL process cancel_request operation, the invocation of that operation and the diversion of the process flow.

Following, the cancel_request scope is analyzed in a lower level.



A message from the Client Partner Link is received, that invokes the cancel_request BPEL process operation

The input parameters of the cancel_request web service are assigned values from the client input variable.

Next, the cancel_request web service is invoked through the respective Partner Link.

The input parameters of the Security Agent are assigned values from the process input variable, the cancel_request web service output variable and a predefined value that defines the Agent behavior that will be executed.

Next, the Security Agent is invoked through the respective Partner Link.

The input parameters of the Workflow Agent are assigned values from the process input variable, the cancel_request web service output variable and a predefined value that defines the Agent behavior that will be executed.

Next, the workflow Agent is invoked through the respective Partner Link.

The output parameter of the delegate BPEL process operation is initialized

A reply is returned to the Client through the respective Partner Link.

The process instance is terminated.

Figure 30. Cancel_Request Scope

8

Layer Comunication

This chapter describes the technologies used for the layer comunication.

This chapter includes the following sections:

- Section 8.1, " Business process Layer Communication "
- Section 8.2, " Business Functionality Layer Communication "
- Section 8.3, " Security And Workflow Layer Communication "
- Section 8.4, " Presentation Layer Communication "

8. 1 BUSINESS PROCESS LAYER COMMUNICATION

The process layer communicates with the other layers, exchanging data, invoking operations and receiving invocations from clients.

8.2 BUSINESS FUNCTIONALITY LAYER COMMUNICATION

BPEL processes interact with external web services in two ways:

• The BPEL process invokes operations on other web services.

• Web services that have been invoked by the BPEL process make callbacks to return replies.

Links to all parties BPEL interacts with are called **partner links**. Partner links can be links to web services that are invoked by the BPEL process.

The process invokes other web services using the <invoke> activity, where it has to specify the operation name and the port type used for invocation. BPEL business processes model the exchange of messages between involved web services. Messages are exchanged as

operations are invoked. Later the service (partner) invokes the callback operation on the process to return the requested data.

To sum up, partner links describe links to partners, where partners might be:

- Services invoked by the process
- Services that invoke the process
- Services that have both roles—they are invoked by the process and they invoke the process

Partner link types allow us to model Service-process relationships as a third party. A partner link type must have at least one role and can have at most two roles. The latter is the usual case. For each role we must specify a portType that is used for interaction.

The partner link types are not part of the BPEL process specification document. Partner link types belong to the service specification and not the process specification. They can therefore be placed in the WSDL document that describes the partner web service or the BPEL process. Partner link types use the WSDL extensibility mechanism, so they can be a part of a WSDL document.

WEB SERVICE PARTNER LINK TYPE

```
<plnk:partnerLinkType name="IssueService">
  <plnk:role name="IssueProvider">
    <plnk:portType name="tns:HelloIF"/>
  </plnk:role>
</plnk:partnerLinkType>
```

BPEL PROCESS PARTNER LINK TYPE: the CheckoutFlow partnerLinkType binds the provider and requester portType into an asynchronous conversation.

```
<plnk:partnerLinkType name="CheckoutFlow">
  <plnk:role name="CheckoutFlowProvider">
    <plnk:portType name="tns:CheckoutFlow"/>
  </plnk:role>
  <plnk:role name="CheckoutFlowRequester">
    <plnk:portType name="tns:CheckoutFlowCallback"/>
  </plnk:role>
</plnk:partnerLinkType>
</definitions>
```

8.3 SECURITY AND WORKFLOW LAYER COMMUNICATION

When the business process reaches a point where security or workflow reconfiguration is required due to alterations to its context, it has to communicate with the security or workflow agent. The security or workflow agent has to be notified that there is work for it to do. It has to be informed about the type of function that has to be executed and about the context parameters it has to manipulate.

For all the above to be accomplished, a java class is used, which communicates with the BPEL workflow through WSIF. This java class receives from the workflow, the context information with the use of XML façades. XML façades help to map data between XML and JAVA classes.

This class is also used, in order to create the JADE run-time environment and to start a new agent that will send a message to the security agent. The message will contain the context information.

```
public AgentController startMessageAgent(String host, // JADE  Main Container host
                                String port, // JADE  Main Container port
                                String name // Message agent name
                               ) {
   // Retrieve the singleton instance of the JADE Runtime
   Runtime rt = Runtime.instance();
   to=new Object[8];
          to[0]=arg1; to[1]=arg2; to[2]=arg3;

   // Create a main container to host the Book Buyer agent
   Profile p = new ProfileImpl();
   p.setParameter(Profile.MAIN_HOST, host);
   p.setParameter(Profile.MAIN_PORT, port);
   ContainerController cc = rt.createAgentContainer(p); //requires a main-container to be already
active
   if (cc != null) {
     // Create the Message agent and start it
     try {
       AgentController ac = cc.createNewAgent(name, "examples.myagents.SendMessageAgent1", to);
       ac.start();
       return ac;
     }
     catch (Exception e) {
       e.printStackTrace();
     }
   }
   return null;
 }
```

The agent that is created has only one behaviour which is a ‘One-shot’ behaviour. ‘One-shot’ behaviours are designed to complete in one execution phase; their action()
method is thus executed only once.

When the message agent is created and started by the above mentioned class, it forms a message for the security or workflow agent. This message defines the behaviour of the security or workflow agent that will be executed and includes the context information. After the message agent sends the message, the message agent is destroyed.

```
        addBehaviour(new OneShotBehaviour() {
                        public void action() {
                                ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
                                msg.addReceiver(new AID("SA", AID.ISLOCALNAME));
                                msg.setContent(targetid);
                                msg.setSender(new AID(getAID().getLocalName(),
AID.ISLOCALNAME));
                                myAgent.send(msg);
                                myAgent.doDelete();
                                takeDown();
                                try{
                                ContainerController cc = myAgent.getContainerController();
                                cc.kill();
                                System.exit(0);
                }//try
    catch (Exception e) {
    }
       takeDown();
                return;
                }
     });
```

JADE run-time automatically posts messages into a receiver’s private message queue as soon as they arrive. An agent can pick up messages from its message queue by means of the receive() method. This method returns the first message in the message queue (thus causing it to be removed), or null if the message queue is empty, and immediately returns.

The action() method of every behaviour is executed only when a new message is received. the behaviours are ‘blocked’ until a new message is inserted into the agent’s message queue. Then all blocked behaviours become available for execution again so that they have a chance to process the received message.

The use of templates, allows for the right behaviour to be executed for each message that an agent receives.

Here, we use the message agent names to declare the behaviour of the security or workflow agent that we want to be executed.

```
        public void action() {
                MessageTemplate mt = MessageTemplate.MatchSender(new
AID("MessageAgent1",AID.ISLOCALNAME ));
                ACLMessage msg = myAgent.receive(mt);

.
.
.
}
```

8.4 PRESENTATION LAYER COMMUNICATION

BPEL processes interact with business process clients in two ways:

- The BPEL process receives invocations from clients, including the initial invocation that initiates the process instance
- The BPEL process makes callbacks to clients to return replies.

Links to all parties BPEL interacts with are called **partner links**. Partner links are concrete references to services and clients that a BPEL business process interacts with.

A BPEL process has at least one <receive> activity to receive invocations by the client, where it has to specify the operation name and port type used for invocation. It also has at least one <reply> activity to return replies to the client.

BPEL treats clients as partner links for two reasons. The most obvious reason is support for asynchronous interactions. In asynchronous interactions, the process needs to invoke operations on its clients. This is used for modeling asynchronous BPEL processes. Such processes also invoke the callback on the initial caller, as mentioned in the previous section.

The second reason is based on the fact that the BPEL process can offer services. These services, offered through port types, can be used by more than one client. The process may wish to distinguish between different clients and offer them only the functionality they are authorized to use.

The partner link types are not part of the BPEL process specification document. Partner link types belong to the service specification and not the process specification. They can therefore be placed in the WSDL document that describes the partner web service or the

BPEL process. Partner link types use the WSDL extensibility mechanism, so they can be a part of a WSDL document.

BPEL PROCESS PARTNER LINK TYPE: The CheckoutFlow partnerLinkType binds the provider and requester portType into an asynchronous conversation.

```
  <plnk:partnerLinkType name="CheckoutFlow">
    <plnk:role name="CheckoutFlowProvider">
      <plnk:portType name="tns:CheckoutFlow"/>
    </plnk:role>
    <plnk:role name="CheckoutFlowRequester">
      <plnk:portType name="tns:CheckoutFlowCallback"/>
    </plnk:role>
  </plnk:partnerLinkType>
</definitions>
```

Each BPEL process has a number of operations that can be invoked by a client. These are also declared in the BPEL process WSDL file.

```
  <!-- portType implemented by the CheckoutFlow BPEL process -->
  <portType name="CheckoutFlow">
    <operation name="initiate">
      <input message="tns:CheckoutFlowRequestMessage"/>
      <output message="tns:CheckoutFlowResponseMessage"/>
    </operation>

    <operation name="cancelrequest">
      <input message="tns:CheckoutFlowCancelRequestMessage"/>
      <output message="tns:CheckoutFlowCancelRequestResponseMessage"/>
    </operation>

    <operation name="writeproposal">
      <input message="tns:CheckoutFlowWriteProposalMessage"/>
      <output message="tns:CheckoutFlowWriteProposalResponseMessage"/>
    </operation>

    <operation name="sendproposal">
      <input message="tns:CheckoutFlowSendProposalMessage"/>
      <output message="tns:CheckoutFlowSendProposalResponseMessage"/>
```

```
</operation>

<operation name="cancelproposal">
   <input message="tns:CheckoutFlowCancelProposalMessage"/>
   <output message="tns:CheckoutFlowCancelProposalResponseMessage"/>
</operation>

<operation name="sendrequest">
   <input message="tns:CheckoutFlowSendRequestMessage"/>
   <output message="tns:CheckoutFlowSendRequestResponseMessage"/>
</operation>

<operation name="bind">
   <input message="tns:CheckoutFlowBindRequestMessage"/>
   <output message="tns:CheckoutFlowBindRequestResponseMessage"/>
</operation>

<operation name="assign">
   <input message="tns:CheckoutFlowAssignRequestMessage"/>
   <output message="tns:CheckoutFlowAssignRequestResponseMessage"/>
</operation>

<operation name="delegate">
   <input message="tns:CheckoutFlowDelegateRequestMessage"/>
   <output message="tns:CheckoutFlowDelegateRequestResponseMessage"/>
</operation>

<operation name="deletereport">
   <input message="tns:CheckoutFlowDeleteReportMessage"/>
   <output message="tns:CheckoutFlowDeleteReportResponseMessage"/>
</operation>

<operation name="sendreport">
   <input message="tns:CheckoutFlowSendReportMessage"/>
   <output message="tns:CheckoutFlowSendReportResponseMessage"/>
</operation>

<operation name="unbind">
   <input message="tns:CheckoutFlowUnbindRequestMessage"/>
   <output message="tns:CheckoutFlowUnbindRequestResponseMessage"/>
```

```
    </operation>

    <operation name="writereport">
      <input message="tns:CheckoutFlowWriteReportMessage"/>
      <output message="tns:CheckoutFlowWriteReportResponseMessage"/>
    </operation>

    <operation name="continue">
      <input message="tns:CheckoutFlowContinueRequestMessage"/>
      <output message="tns:CheckoutFlowContinueResponseMessage"/>
    </operation>
    <operation name="submit">
      <input message="tns:CheckoutFlowSubmitRequestMessage"/>
      <output message="tns:CheckoutFlowSubmitResponseMessage"/>
    </operation>
  </portType>

  <!-- portType implemented by the requester of CheckoutFlow BPEL process
       for asynchronous callback purposes
       -->
  <portType name="CheckoutFlowCallback">
    <operation name="onResult">
      <input message="tns:CheckoutFlowResultMessage"/>
    </operation>
  </portType>
```

To initiate a business process instance, the client first generates the ConversationId, through which he will be able to identify each process instance. The ConversationId is stored in the request record, along with the requestid.

```
VMID guid = new VMID();
String conversationId = guid.toString();
```

Every time a client wishes to interact with the BPEL process, he has to format the data he wants to send to the process. As we have already mentioned, because BPEL variables are XML and Java variables are not, we need a mapping between XML and Java. To handle XML data from Java we use XML façades. XML façades are a set of Java interfaces and

classes through which we can access and modify XML data stored in BPEL variables using get/set methods.

```
OrderType order = OrderTypeFactory.createFacade(new QName("http://samples.otn.com",
"order"));

order.setConversationId(resp);
order.setCroId(request.getParameter("id"));
order.setCustomerId(request.getParameter("custid"));
order.setEmployeeId(" ");
order.setInvokeWeb(" ");
order.setInvokeWsif(" ");
order.setMaDelegId(" ");
order.setMaId(" ");
order.setMemo(" ");
order.setProductId(" ");
order.setQuantity(" ");
order.setRequestId(request.getParameter("requestid"));
order.setSpec(request.getParameter("spec"));
order.setWf_S_Agent(" ");
```

A connection with the BPEL process instance has to be established. The Conversationid is used to determine the process instance for a given request. Another important parameter is the process operation that will be invoked.

```
Locator locator = new Locator("default","bpel");
IDeliveryService deliveryService =
(IDeliveryService)locator.lookupService(IDeliveryService.SERVICE_NAME );
HashMap properties = new HashMap();
properties.put("conversationId",resp);
properties.put("title","Checkout Flow");

// construct the normalized message and send to the BPEL process manager
NormalizedMessage nm = new NormalizedMessage( );
nm.addPart("payload", order.getRootElement() );
nm.setProperties(properties );
NormalizedMessage res = deliveryService.request("CheckoutFlow", "initiate", nm);
```

The BPEL process returns the reply to the client, again using XML façades.

```
Map payload = res.getPayload();
Element el =(Element)payload.get("payload");
order = OrderTypeFactory.createFacade( el );

<div style="background-color:#2D517D; border-top:10px solid white;border-bottom:10px solid white;padding:10px">
<b><font color="#FFFFFF">Send Requestid:<%=order.getRequestId()%></font></b>
</div>
```

9

Runtime Demonstration

In this chapter, a business process instance will be created and the user interface, the interactions between the different layers and the flow of the process will be presented.

This chapter includes the following sections:

- Section 9.1, " Issue Request "
- Section 9.2, " Write Report "
- Section 9.3, " Write Proposal "

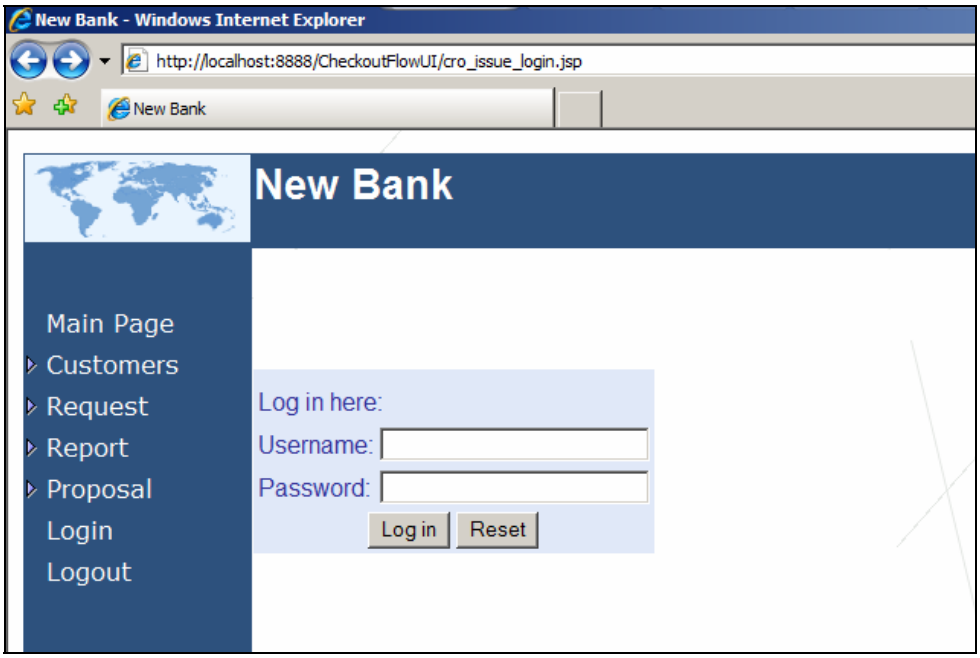
9.1 ISSUE REQUEST

The Issue Request subproccess can be further analyzed in the following activities:

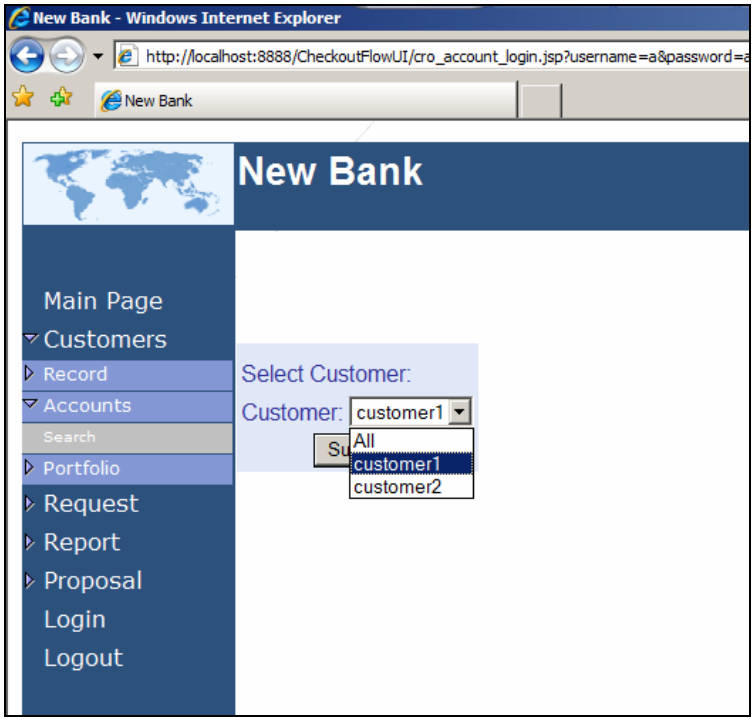
- Access Customer Information
- Issue Request
- Edit Request
- Send Request
- Cancel Request

ACCESS CUSTOMER INFORMATION

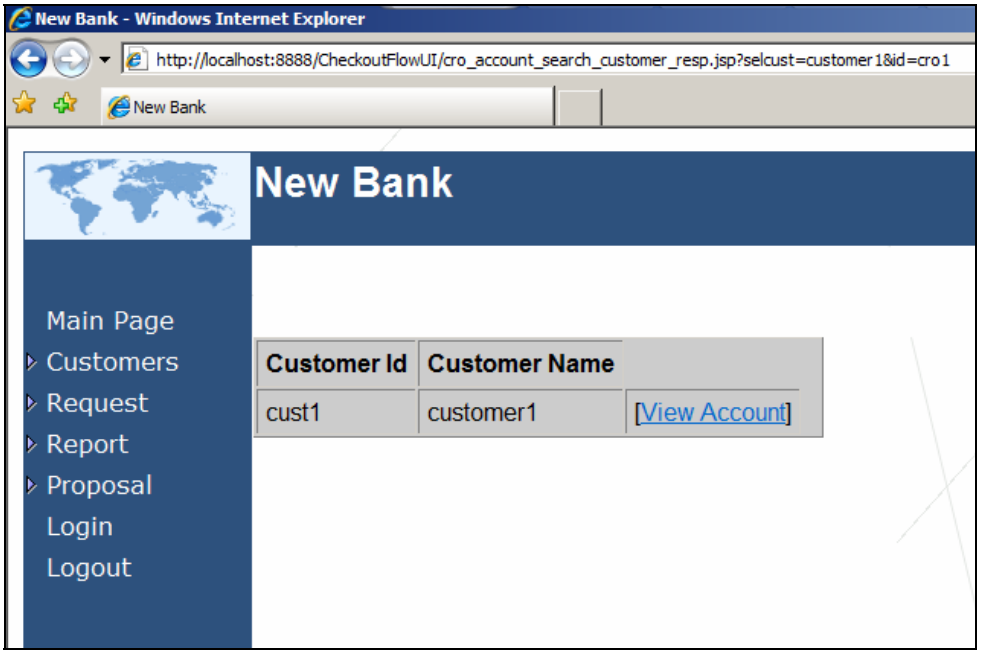
A CRO can access information concerning his customers only



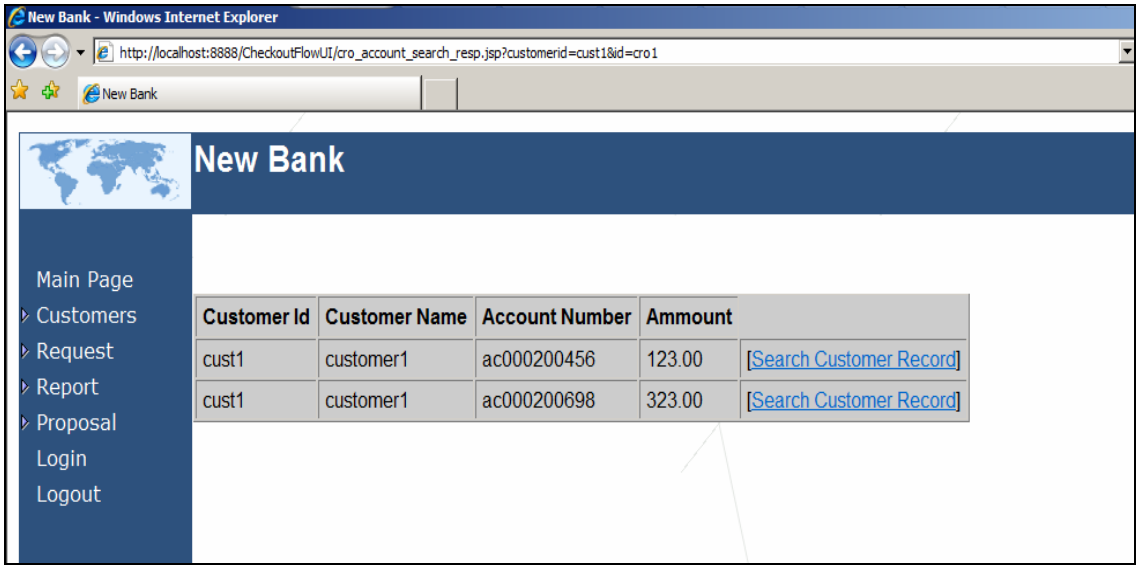
A CRO receives a list of his customers



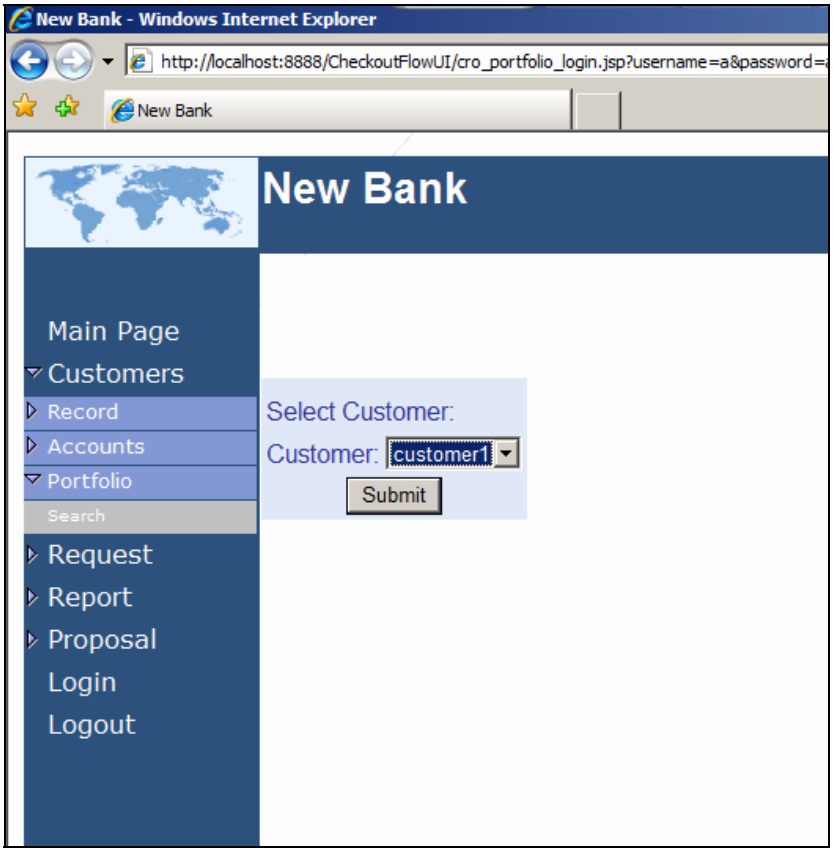
The accounts of these customers can be accessed by the CRO

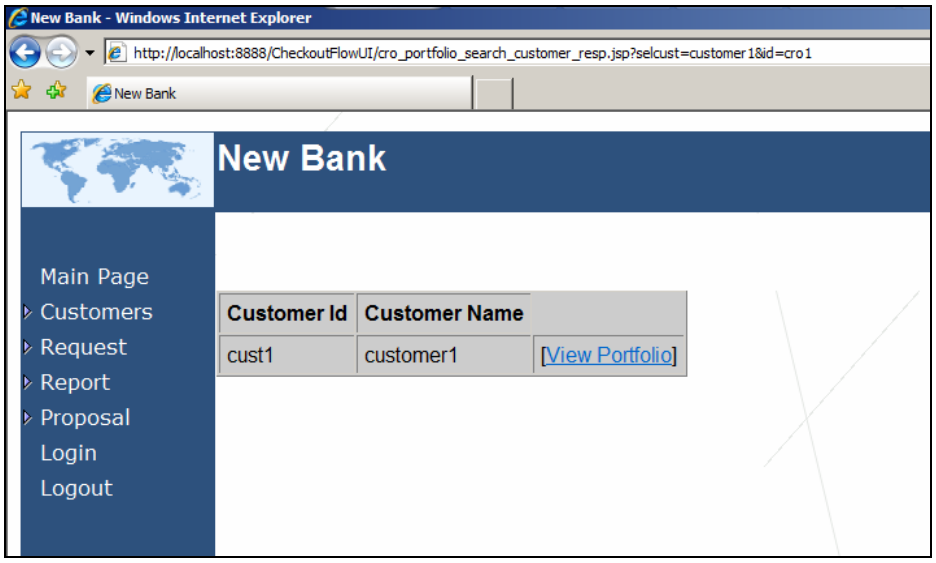


The list of customer accounts is returned

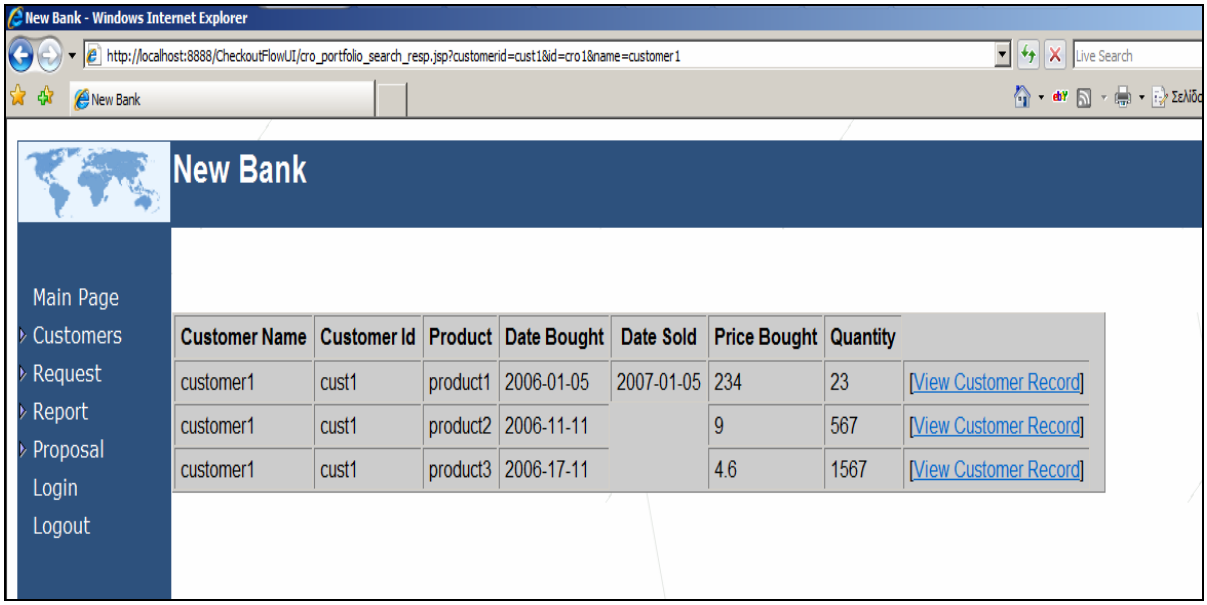


The portfolio of these customers can be accessed by the CRO



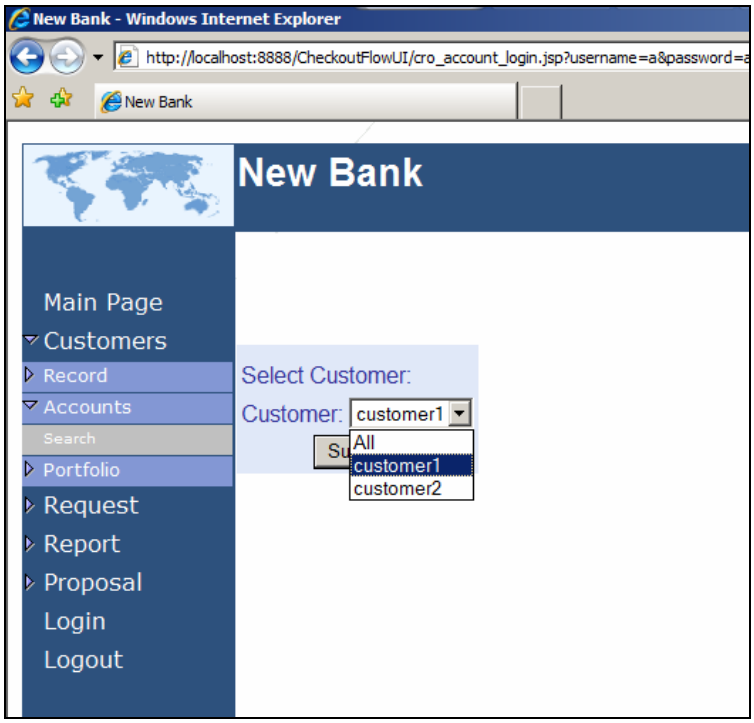


The customer portfolio is returned

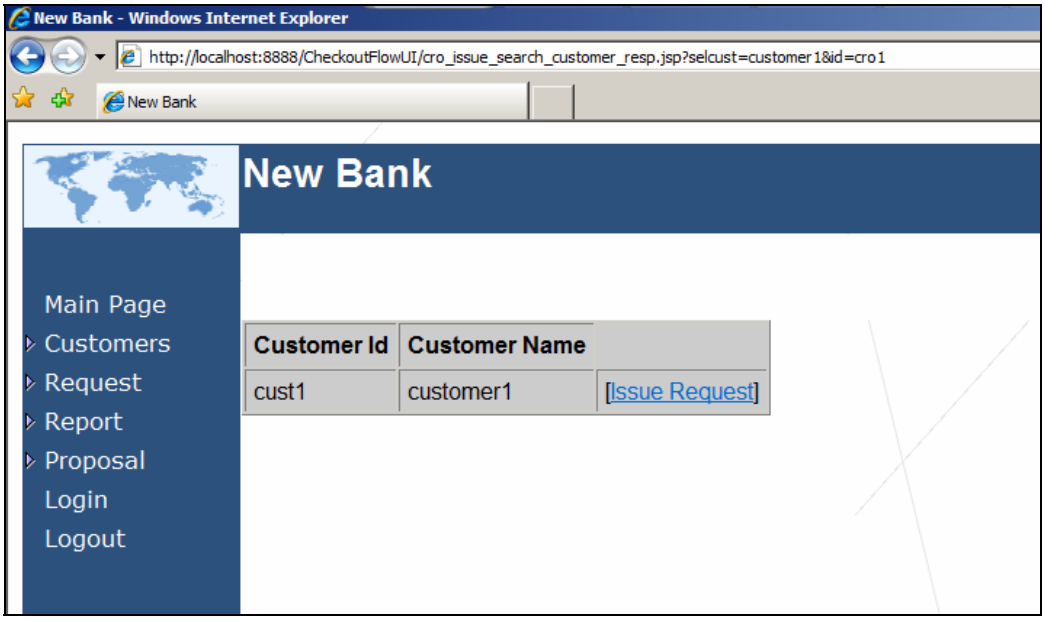


ISSUE REQUEST

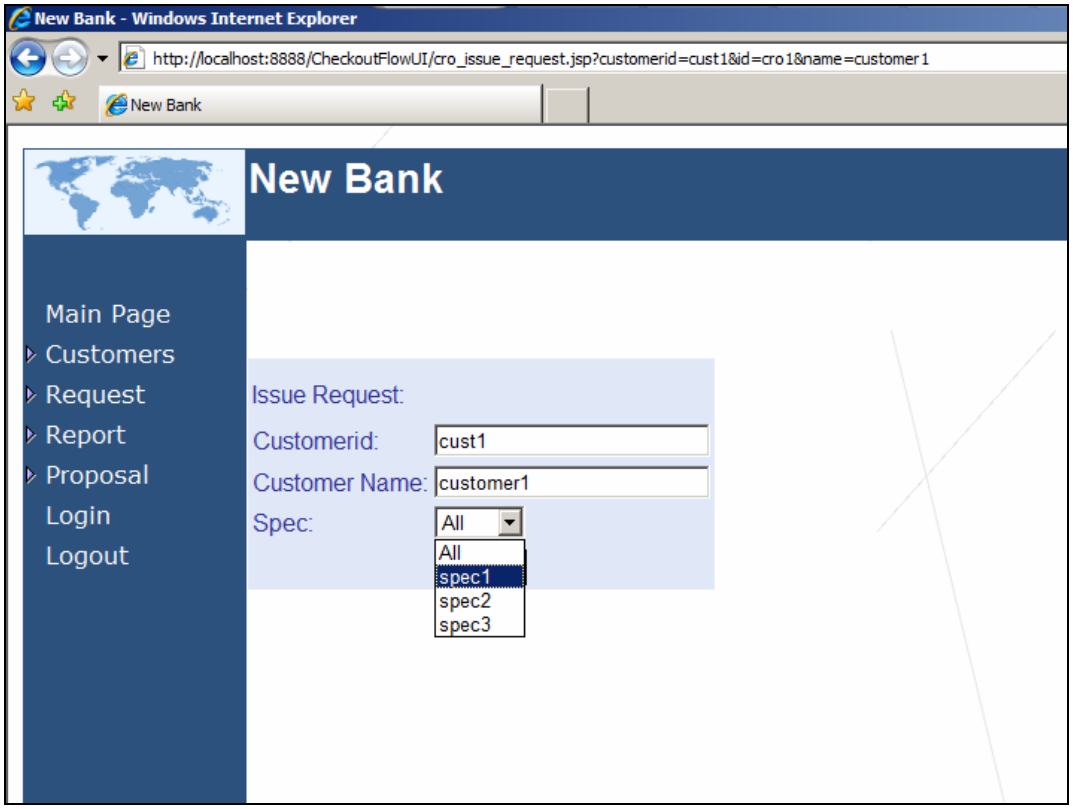
A CRO receives a list of his customers



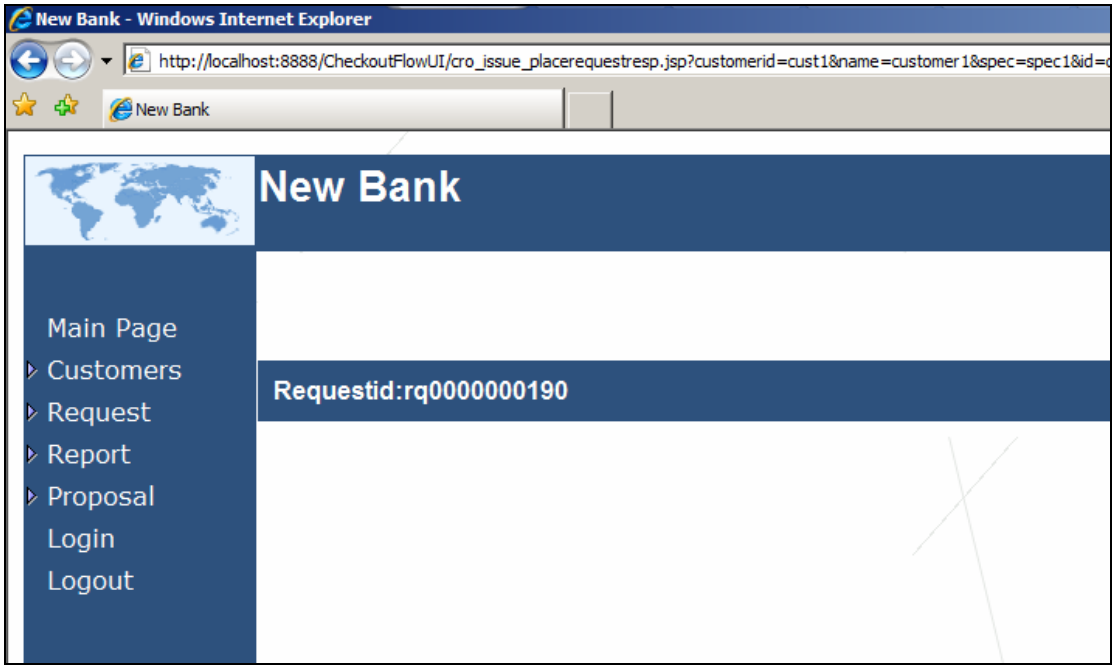
The CRO decides to issue a request for one of his customers



A request is issued for a given customer and specialty



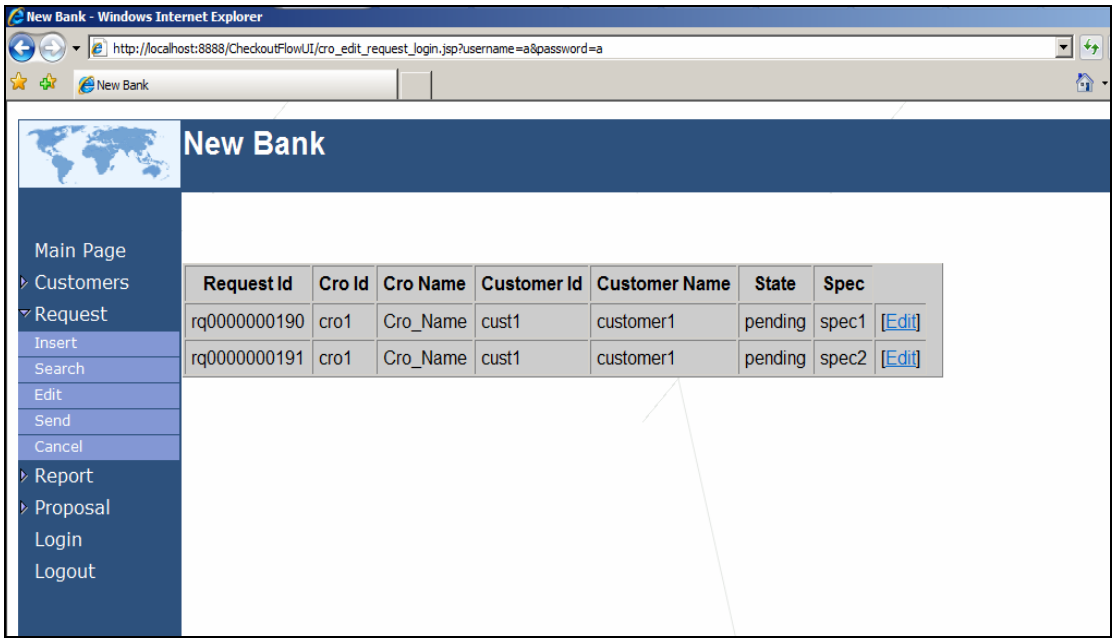
The system returns the ID of the issued request

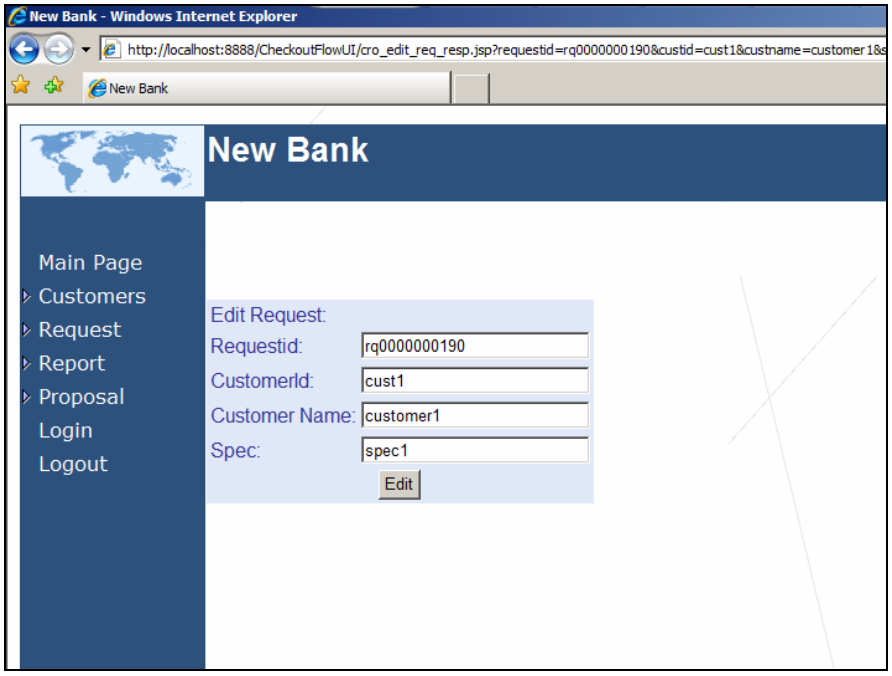


EDIT REQUEST

Before a request is send to the MA department, it can be edited.

A list of the pending requests he issued is returned to the CRO

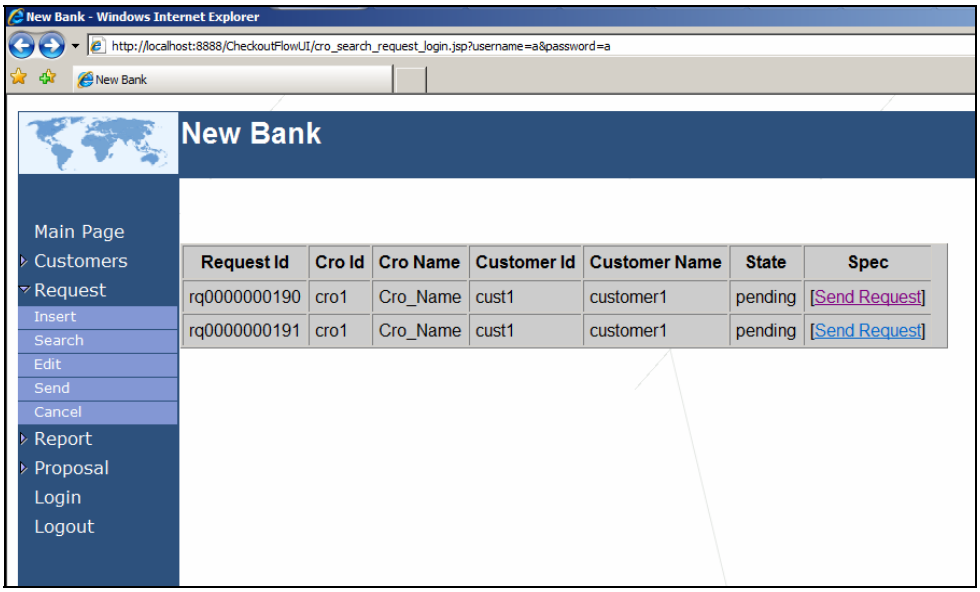




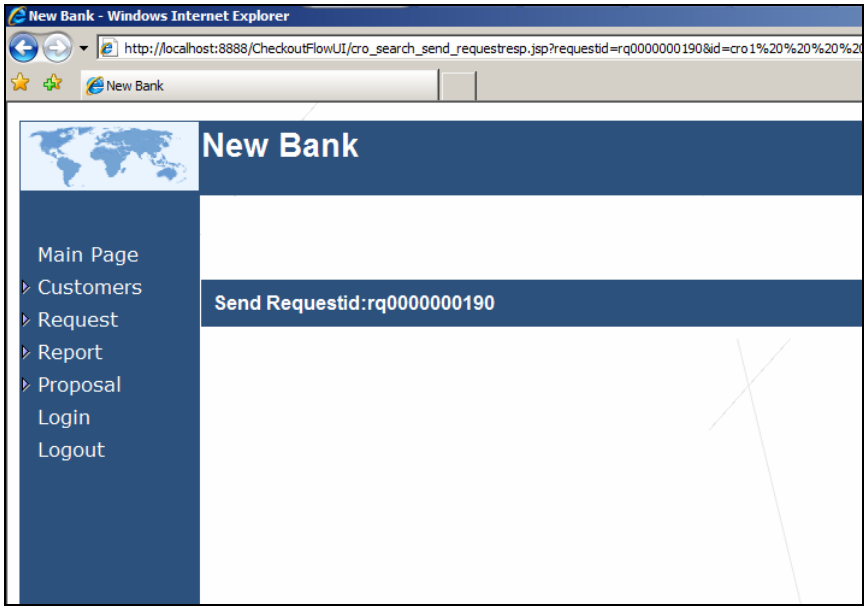
SEND REQUEST

The CRO decides to send the request.

A list of the pending requests he issued is returned to the CRO



The CRO sends a request and the system confirms that the request was send.



When a request is send, a new workflow instance is created. The instance ConversationId, which is used for instance identification, is stored in the database along with the RequestId. The request parameters are passed as input parameters of the workflow.

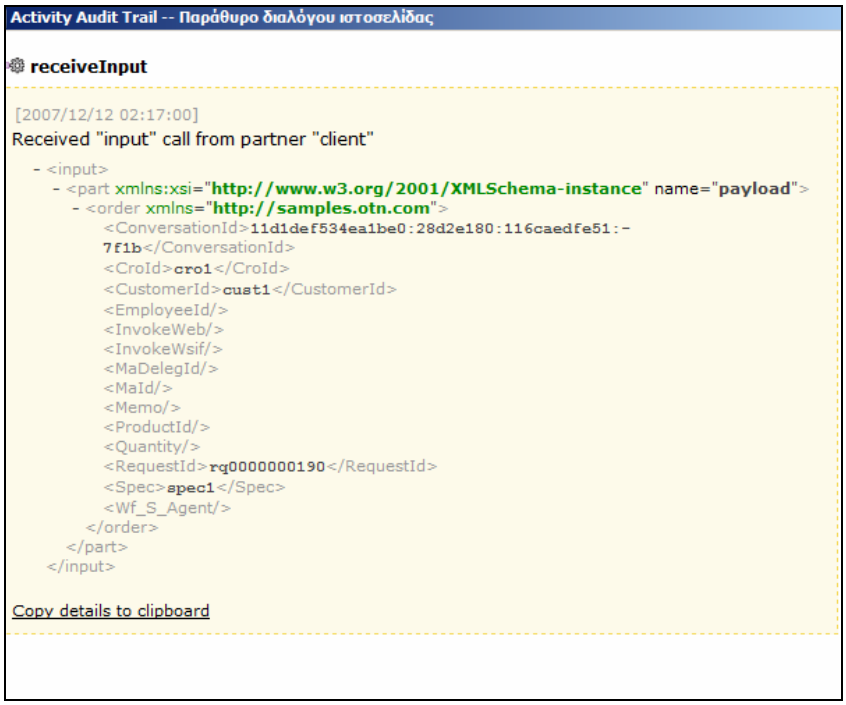


Figure 31. Workflow Instance Input Parameter

First, a web service is invoked by the workflow, to send the request to the MA department. The messages exchanged are the following:

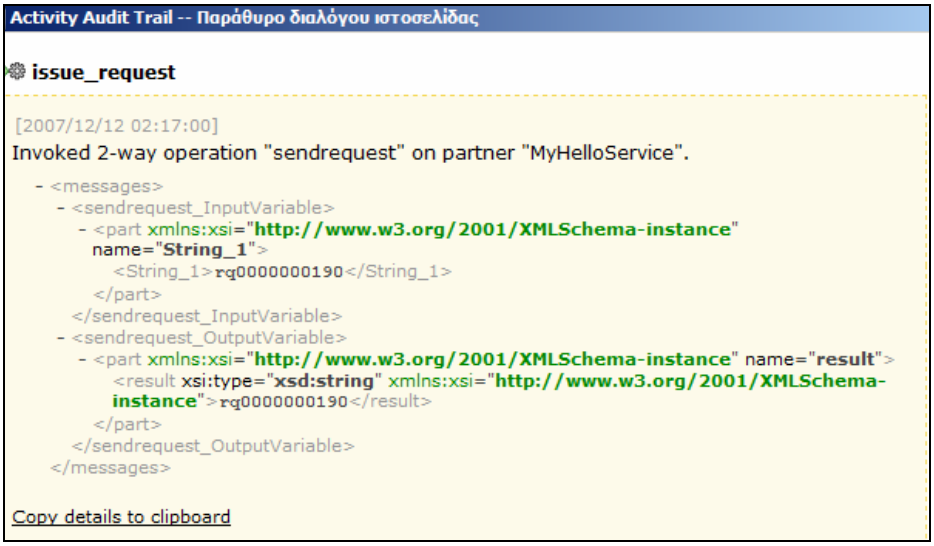


Figure 32. Send_Request Message

Next, the security and workflow agents are invoked, to configure the access rights and the task assignments. The messages exchanged are the following

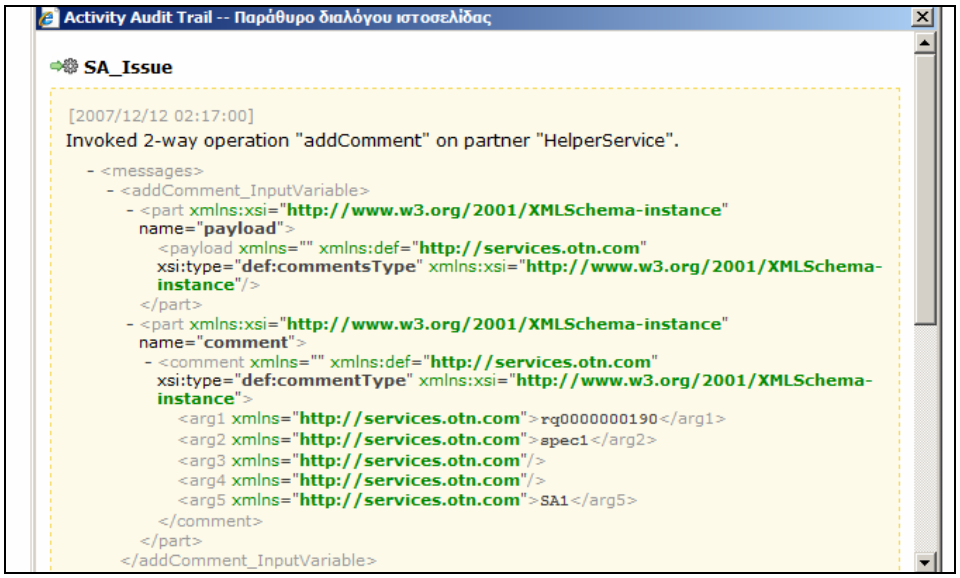


Figure 33. Security Agent Message

Figure 34. Workflow Agent Message

A confirmation message is returned to the CRO who issued the request.

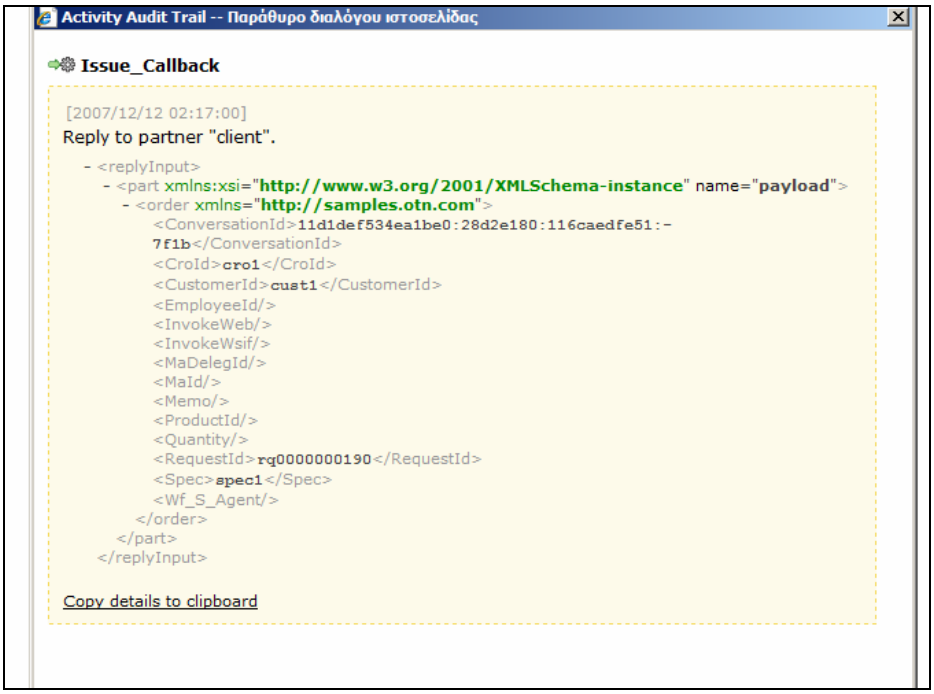


Figure 35. Callback Message

In the Figure 36, the interactions of the client, workflow and services are presented.

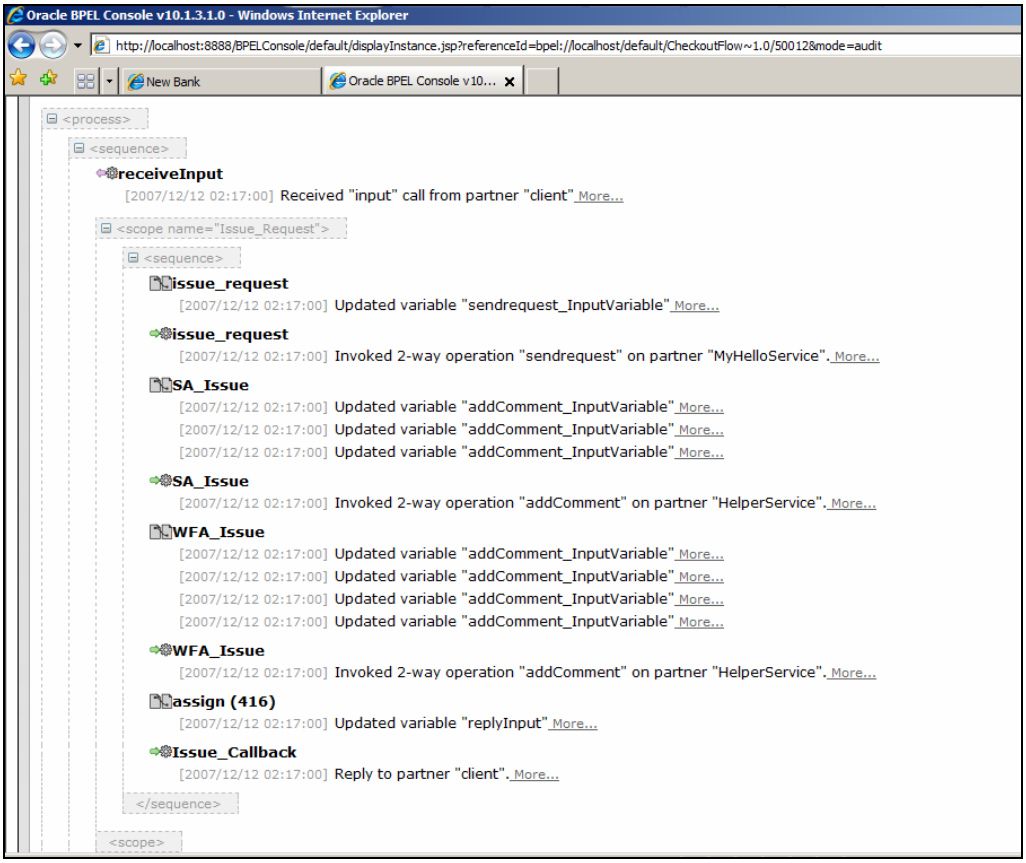


Figure 36. Issue_request Scope Interactions

In Figure 37, the workflow for the Issue_Request scope is presented.

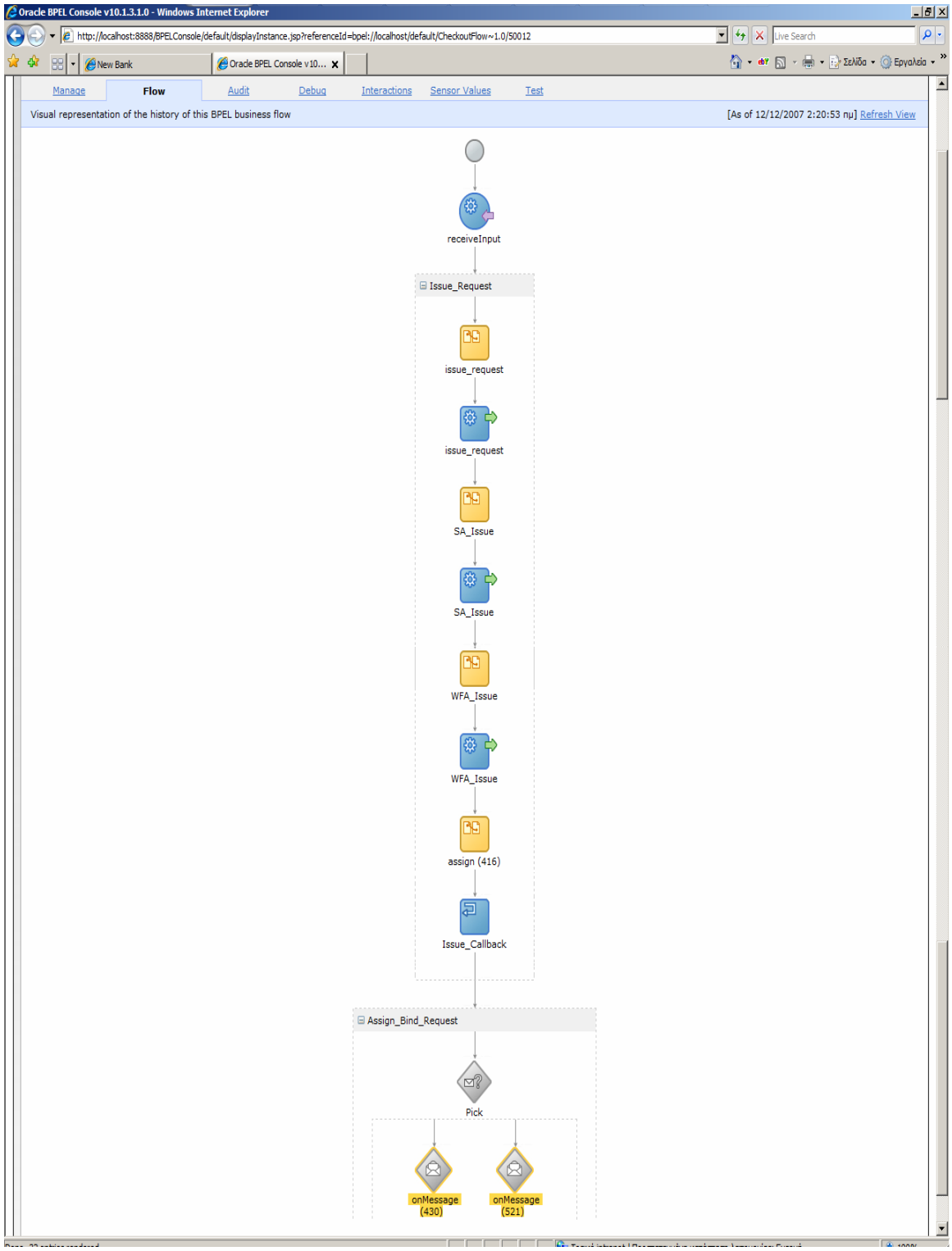


Figure 37. Issue_Request Scope

After the Issue_Request scope, the workflow blocks waiting for an MA to bind the request or the head of MAs to assign it.

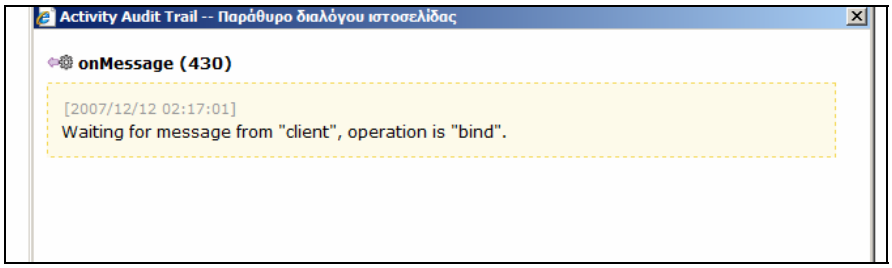


Figure 38. Waiting for a Bind Message

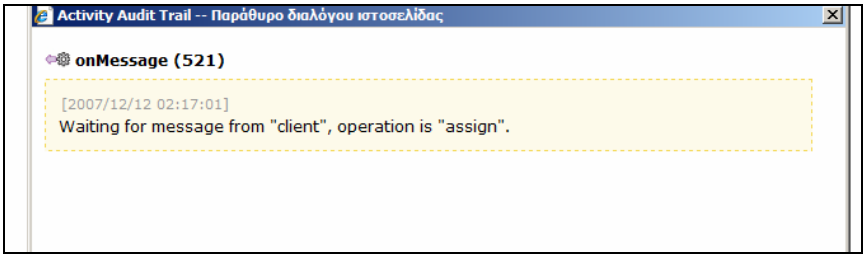


Figure 39. Waiting for an Assign Message

9.2 WRITE REPORT

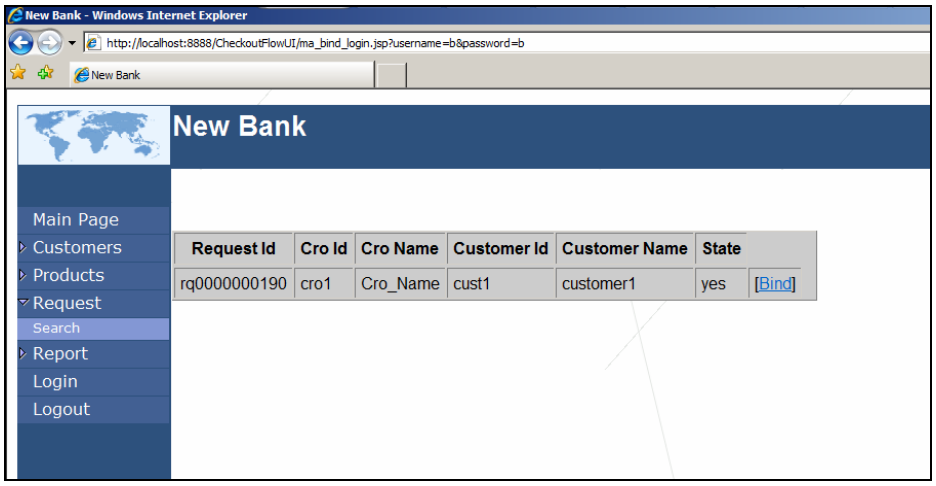
The Write Report subproccess can be further analyzed in the following activities:

- Bind Request
- Access Customer Information
- Delegate Request
- Unbind Request
- Assign Request
- Write Report
- Send Report

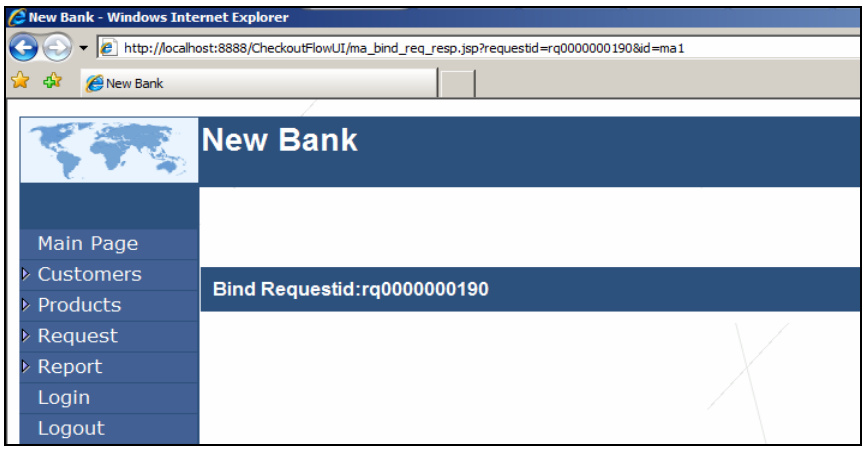
BIND REQUEST

When a MA searches for new requests assigned to him, a list of requests is returned.

A MA decides to bind the previously issued request.



The system returns a confirmation message.



When a MA binds a request, the ConversationId of the respective workflow instance is retrieved from the database and a message is send to that instance. The message describes the workflow operation that will be invoked (the bind operation in this case) and the input parameters.

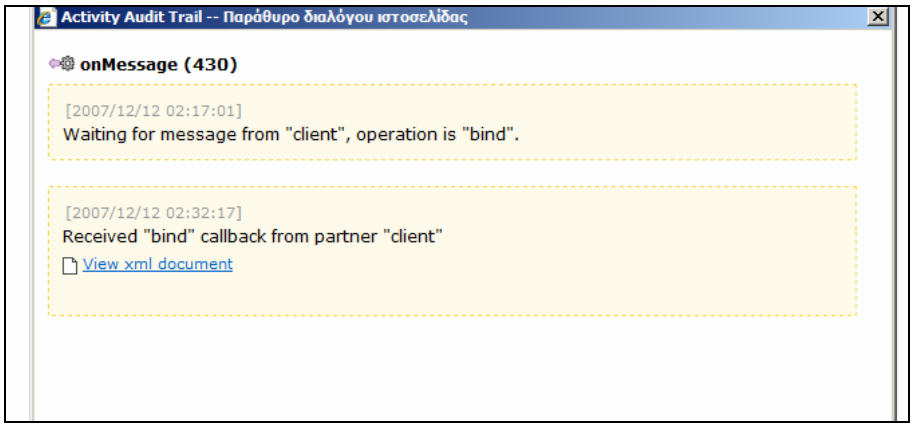


Figure 40. Receive Client Bind Message

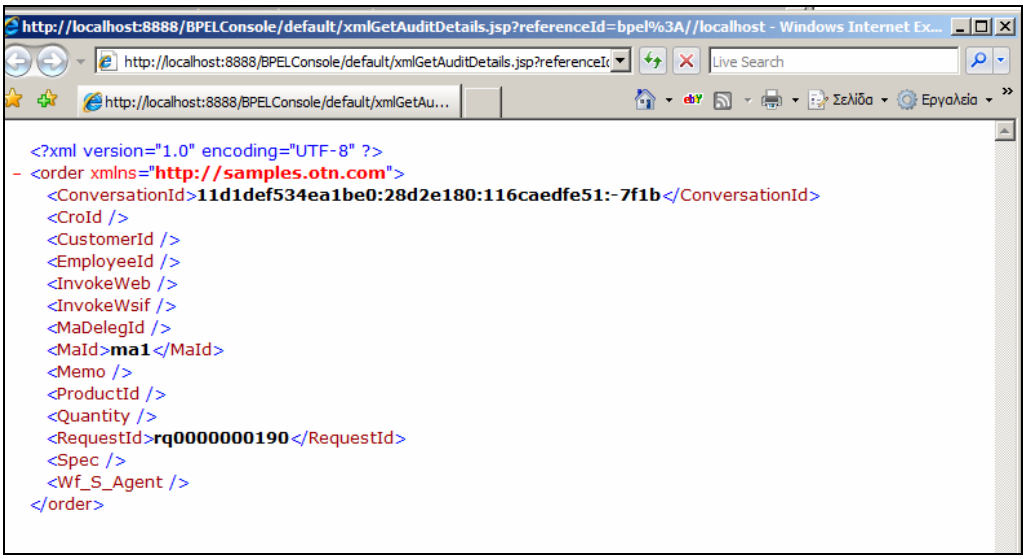


Figure 41. Workflow Instance Bind Message

First, a web service is invoked by the workflow, to bind the request. The messages exchanged are the following:

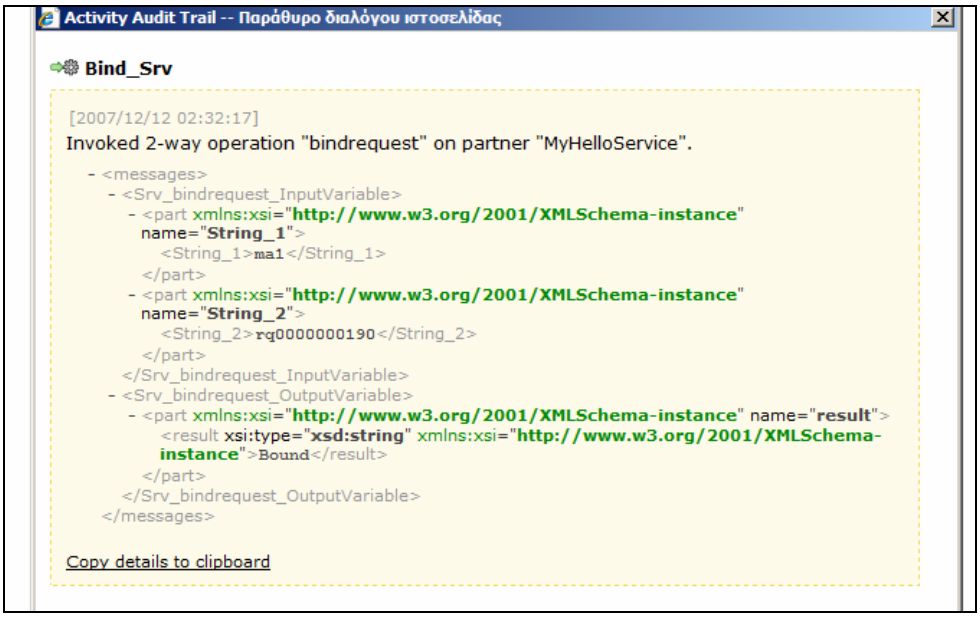


Figure 42. Bind_Request Message

Next, the security and workflow agents are invoked to configure the access rights and the task assignments. The messages exchanged are the following

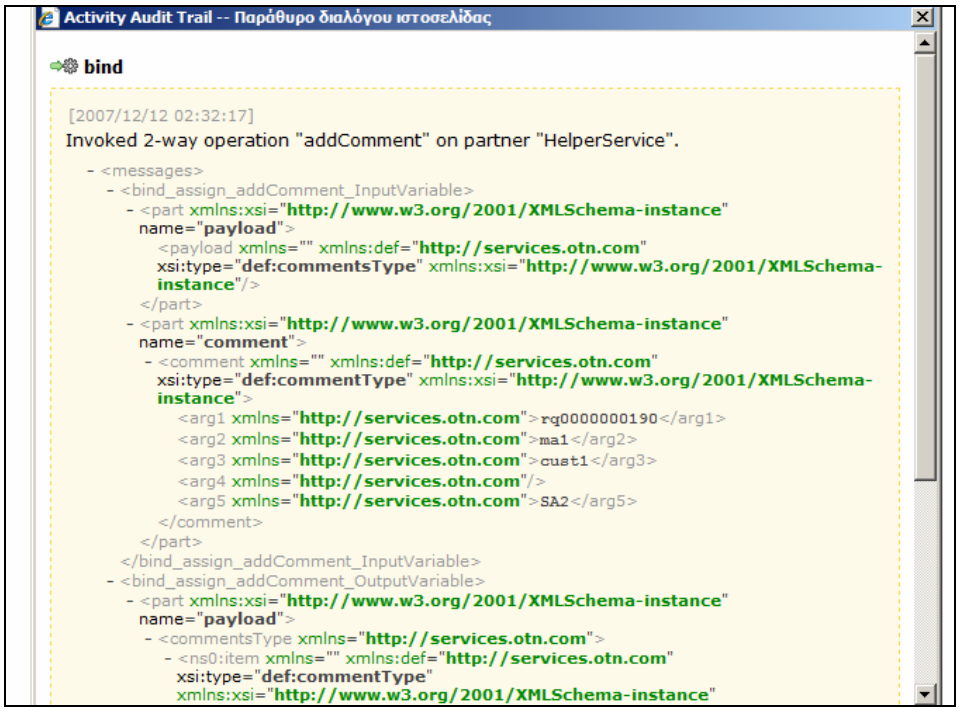


Figure 43. Security Agent Message

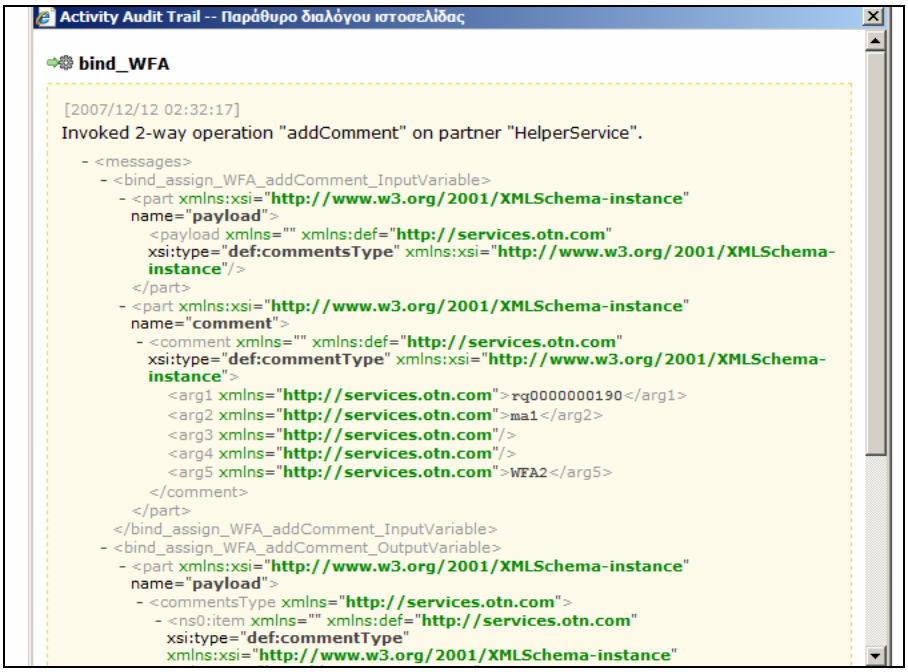


Figure 44. Workflow Agent Message

A confirmation message is returned to the CRO who issued the request.

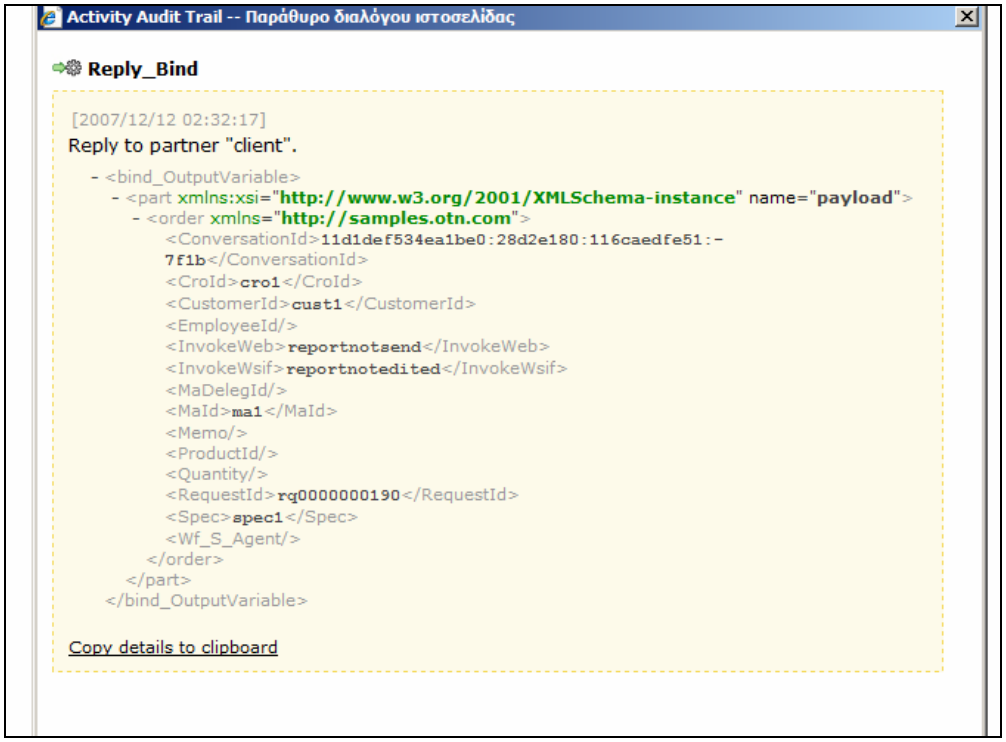


Figure 45. Callback Message

In Figure 46, the interactions of the client, workflow and services are presented.

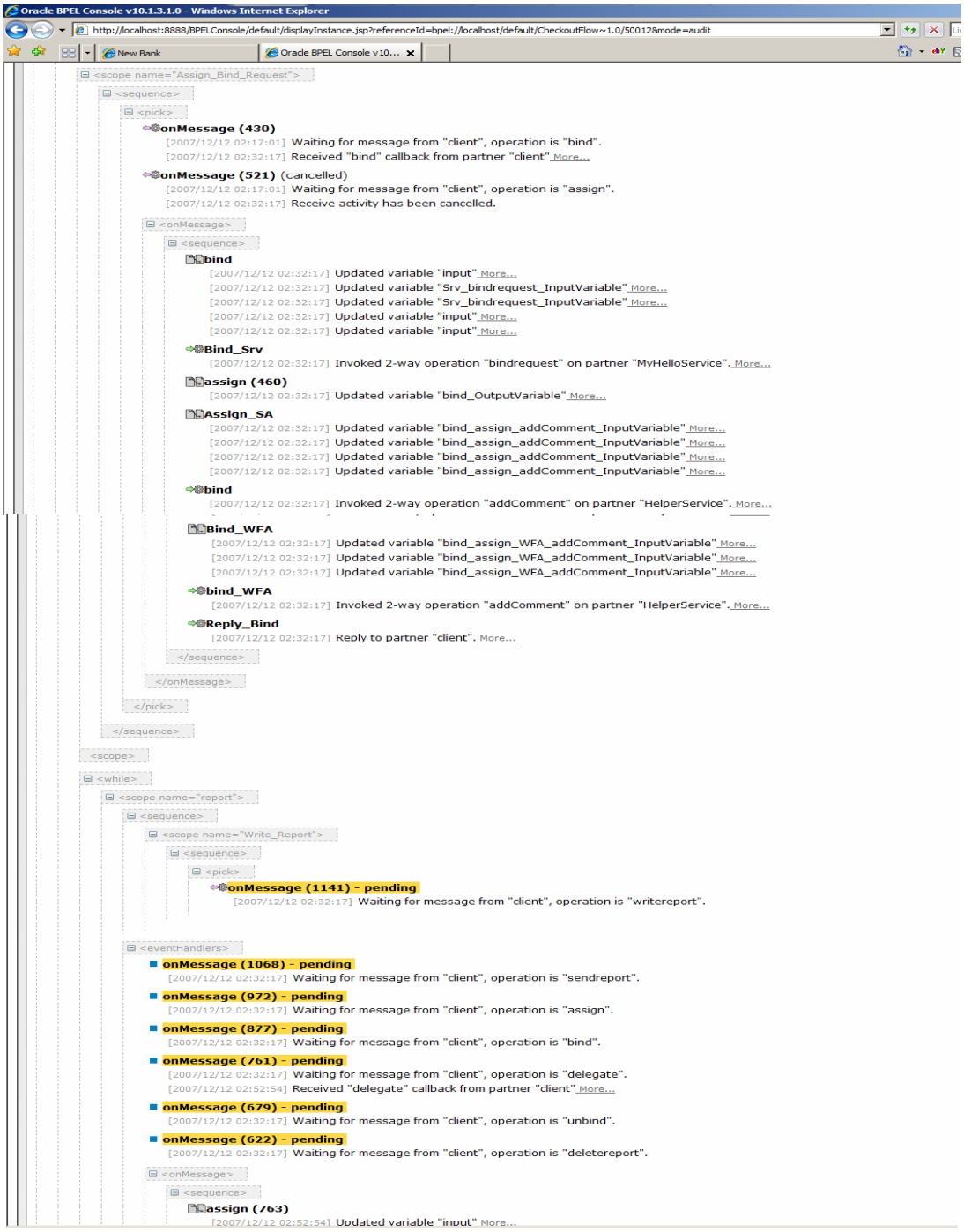


Figure 46. Bind_Request Scope Interactions

In Figure 47, the workflow for the Bind_Request scope is presented.

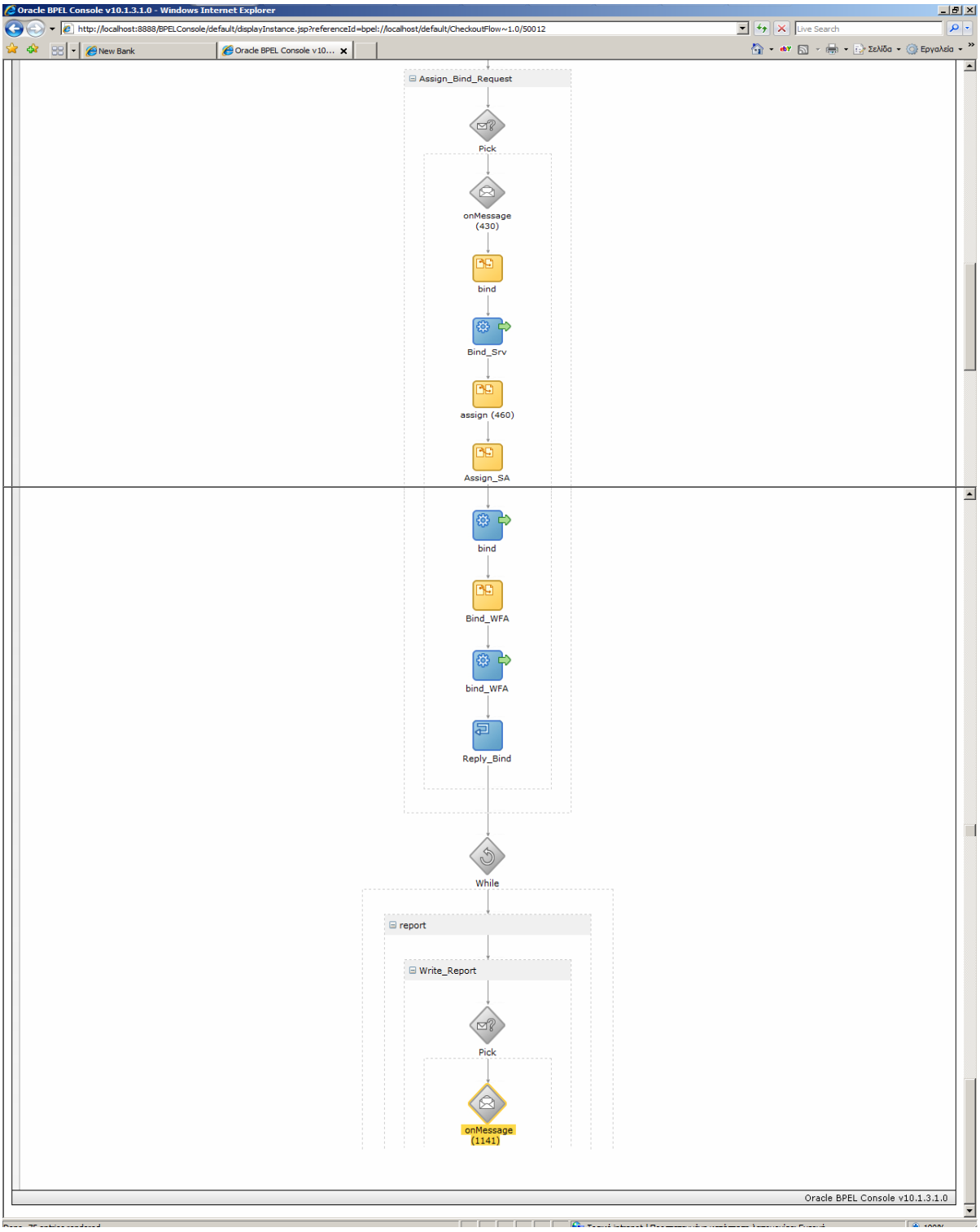


Figure 47. Bind _Request Scope

After the Bind_Request scope, the workflow blocks waiting for an MA to write the report for the request he bound. After the MA binds the request, the process listens for events that will divert the workflow (unbind request, delegate request, assign request, bind request, delete report). Once such an event occurs, predefined operations will be invoked to handle that event.

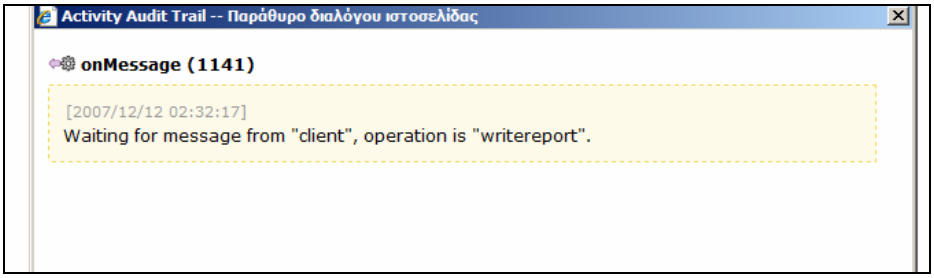
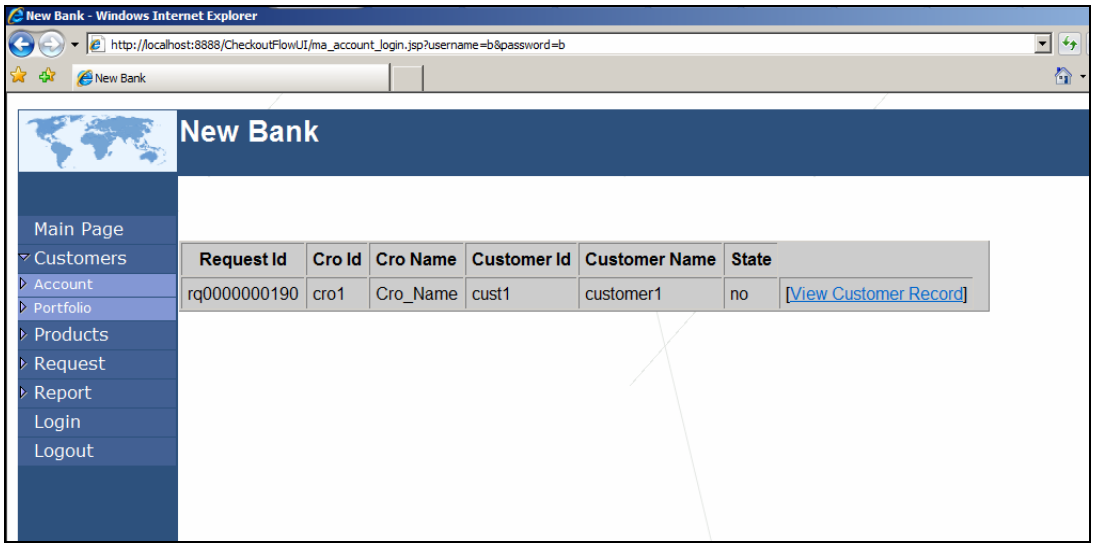


Figure 48. Waiting for a Write_Report Message

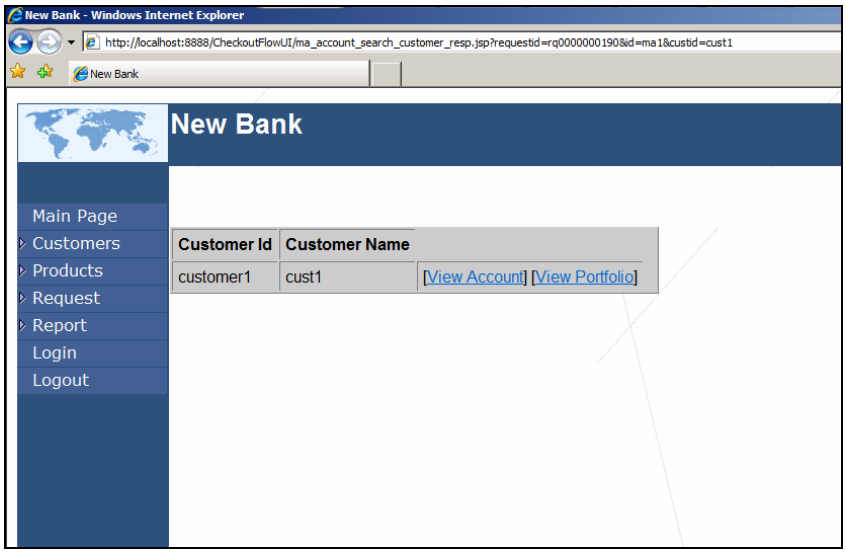
ACCESS CUSTOMER INFORMATION

When a MA binds a request, he has access to the respective customer's information.

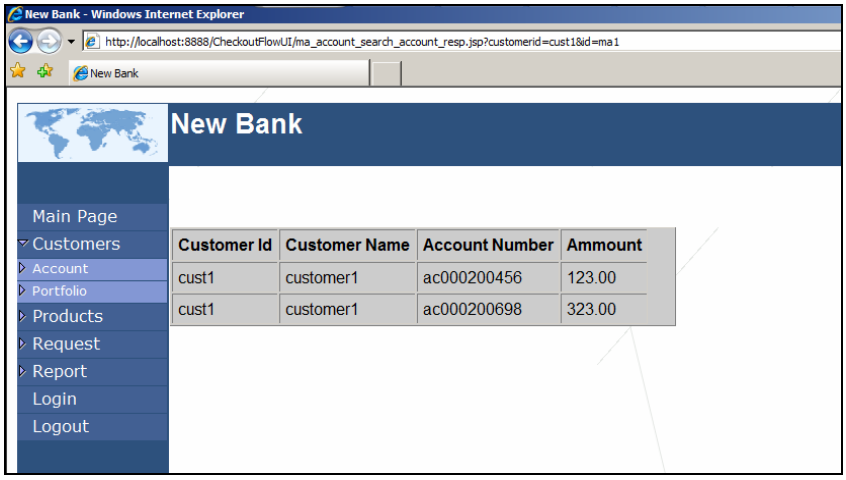
When he searches for customer information, he receives a list of the customers he has bound requests for.



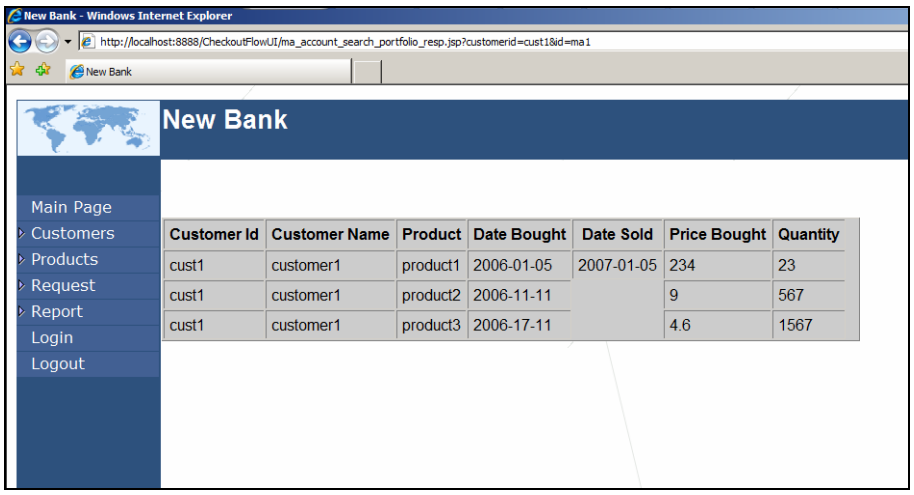
He can view the accounts and the portfolio of these customers.



Customer Accounts



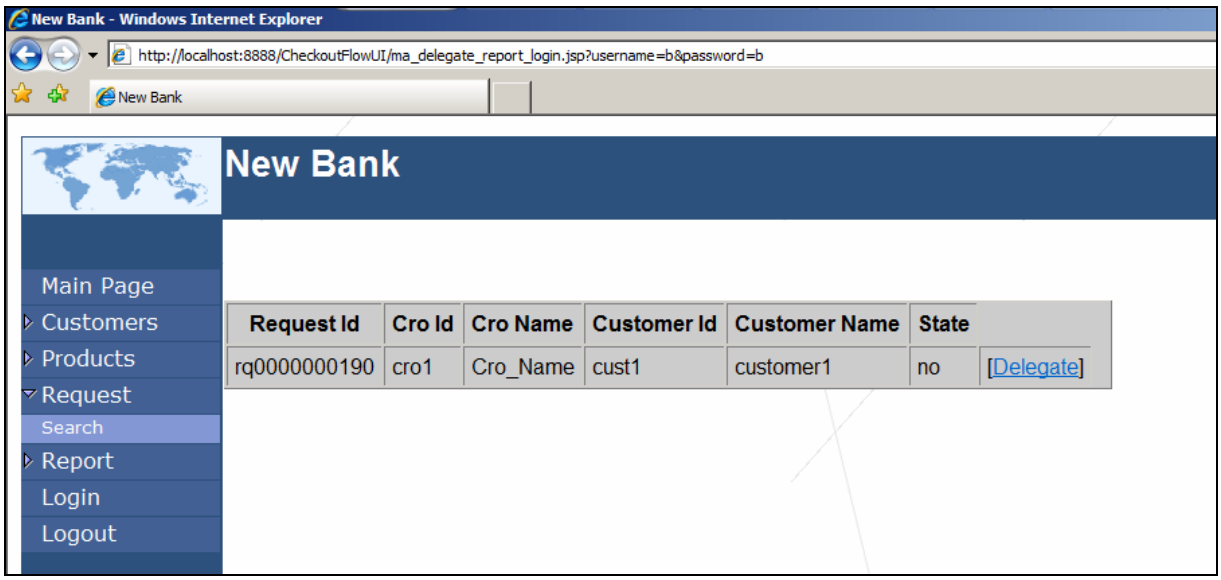
Customer Portfolio



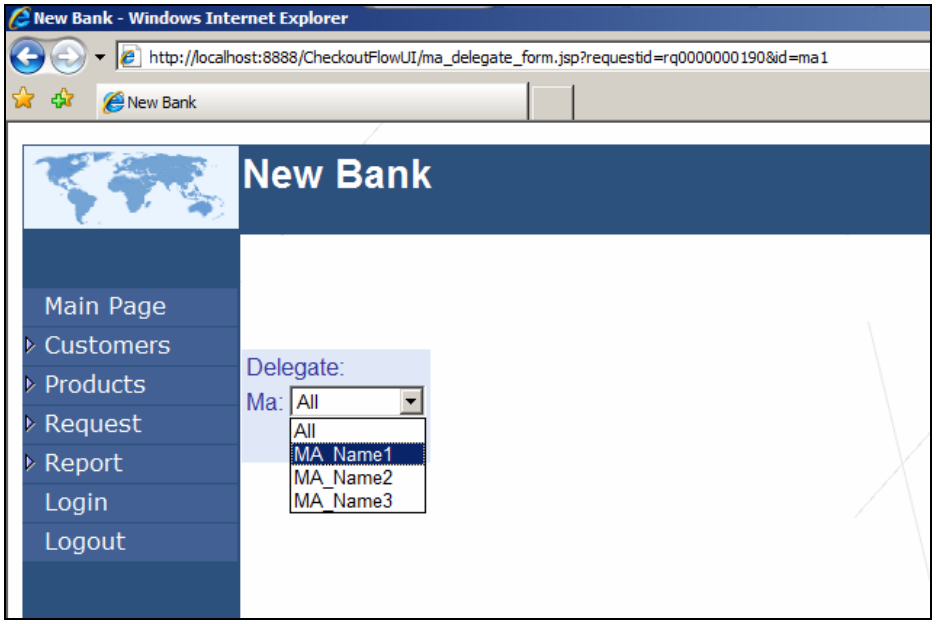
DELEGATE REQUEST

A MA who bound a request, can delegate it to one of the initially qualifying MAs. When a MA delegates a request, the ConversationId of the respective workflow instance is retrieved from the database and a message is send to that instance. The message describes the workflow operation that will be invoked (the delegate operation in this case) and the input parameters.

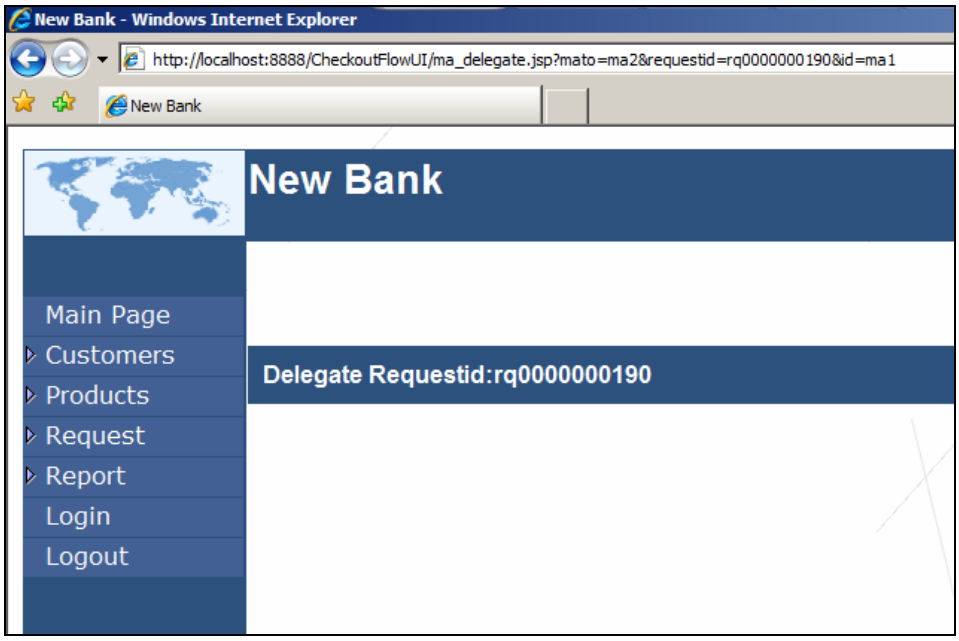
The MA receives a list of the requests, he has bound



When he selects a request, a list of the originally qualifying MAs is returned.



He delegates the request to one of the listed MAs and the system returns a confirmation message.



First, a web service is invoked by the workflow, to delegate the request. Next, the security and workflow agents are invoked, to configure the access rights and the task assignments. Finally, a confirmation message is returned.

In Figure 49, the workflow for the Delegate_Request event is presented.

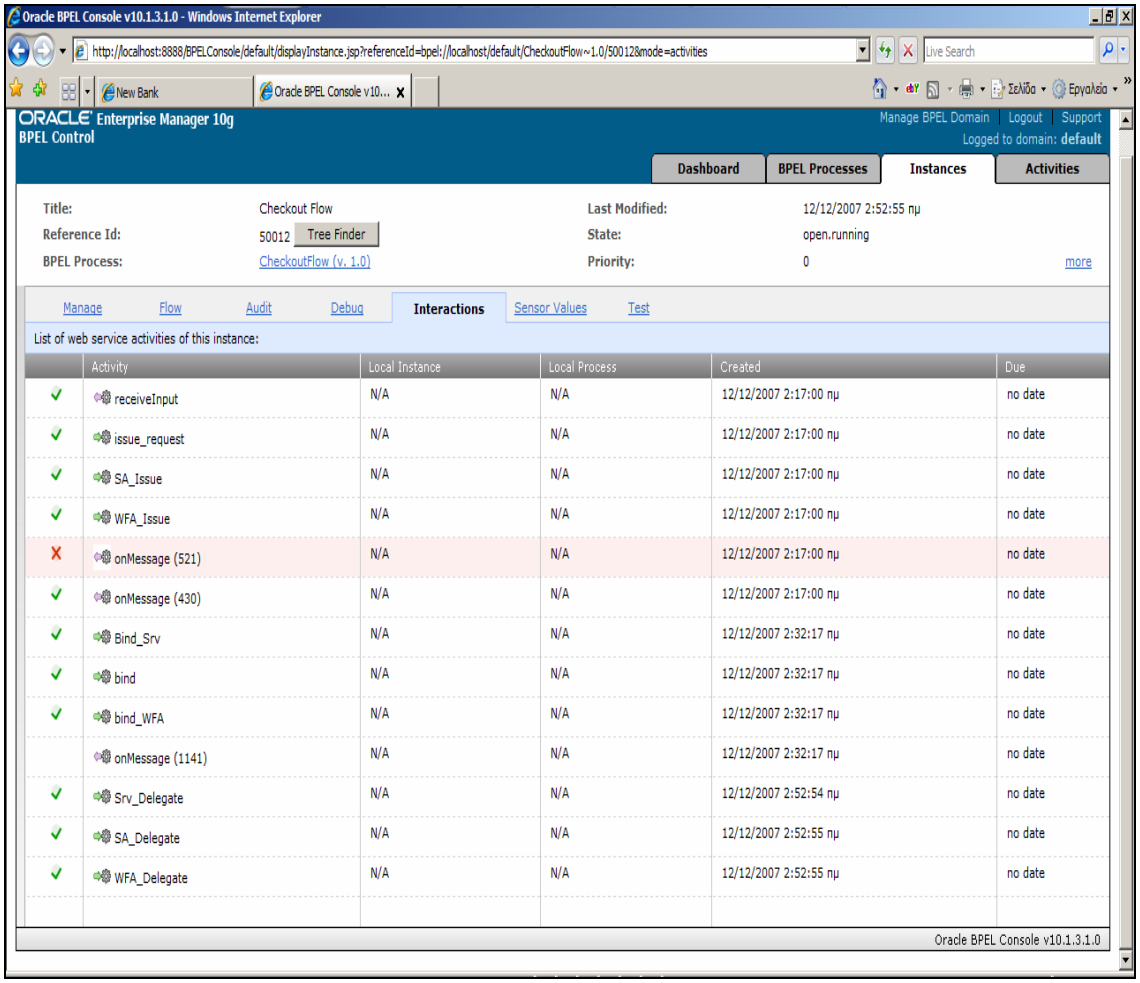


Figure 49. Delegate_Request Event

In Figure 50, the interactions of the client, workflow and services are presented.

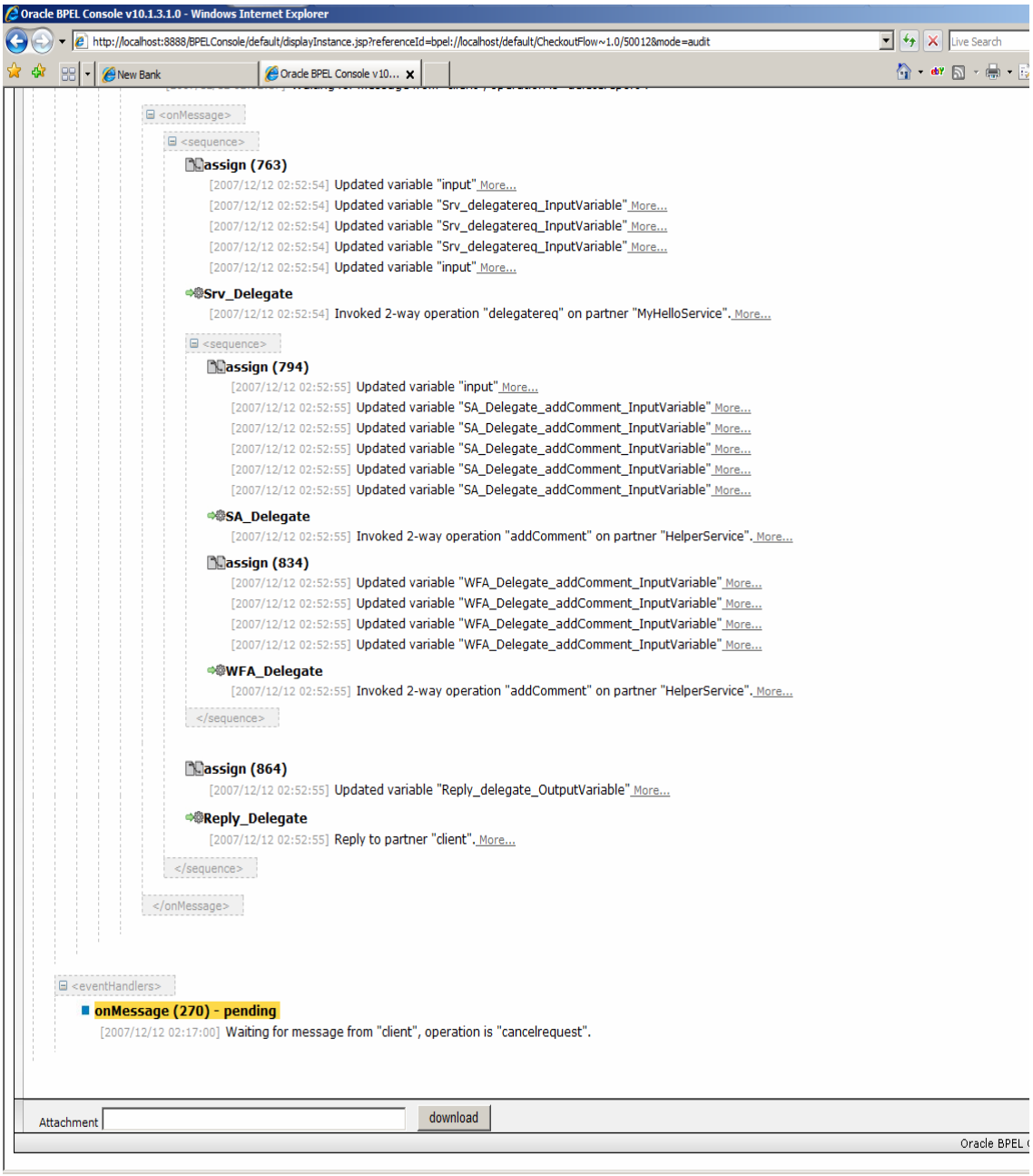
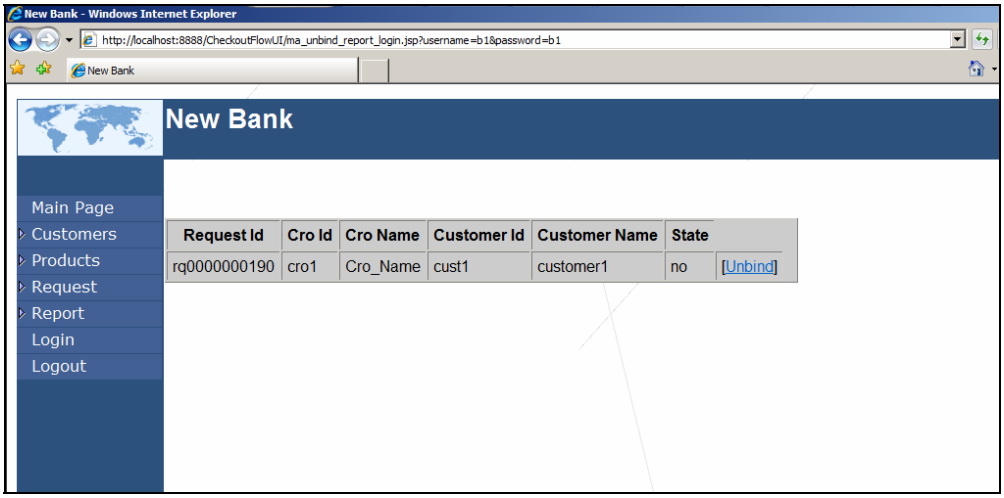


Figure 50. Delegate_Request Event Interactions

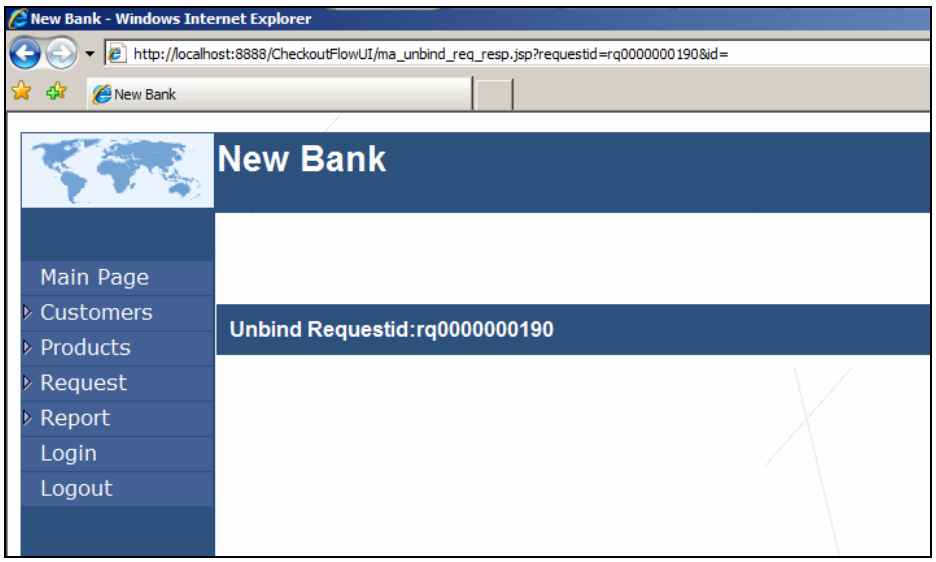
UNBIND REQUEST

A MA who bound a request, can unbind it. When a MA unbinds a request, the ConversationId of the respective workflow instance is retrieved from the database and a message is send to that instance. The message describes the workflow operation that will be invoked (the unbind operation in this case) and the input parameters.

The MA receives a list of the requests, he has bound.



He unbinds the request and the system returns a confirmation message.



First, a web service is invoked by the workflow, to unbind the request. Next, the security and workflow agents are invoked, to configure the access rights and the task assignments. Finally, a confirmation message is returned.

In Figure 51, the workflow for the Unbind_Request event is presented.

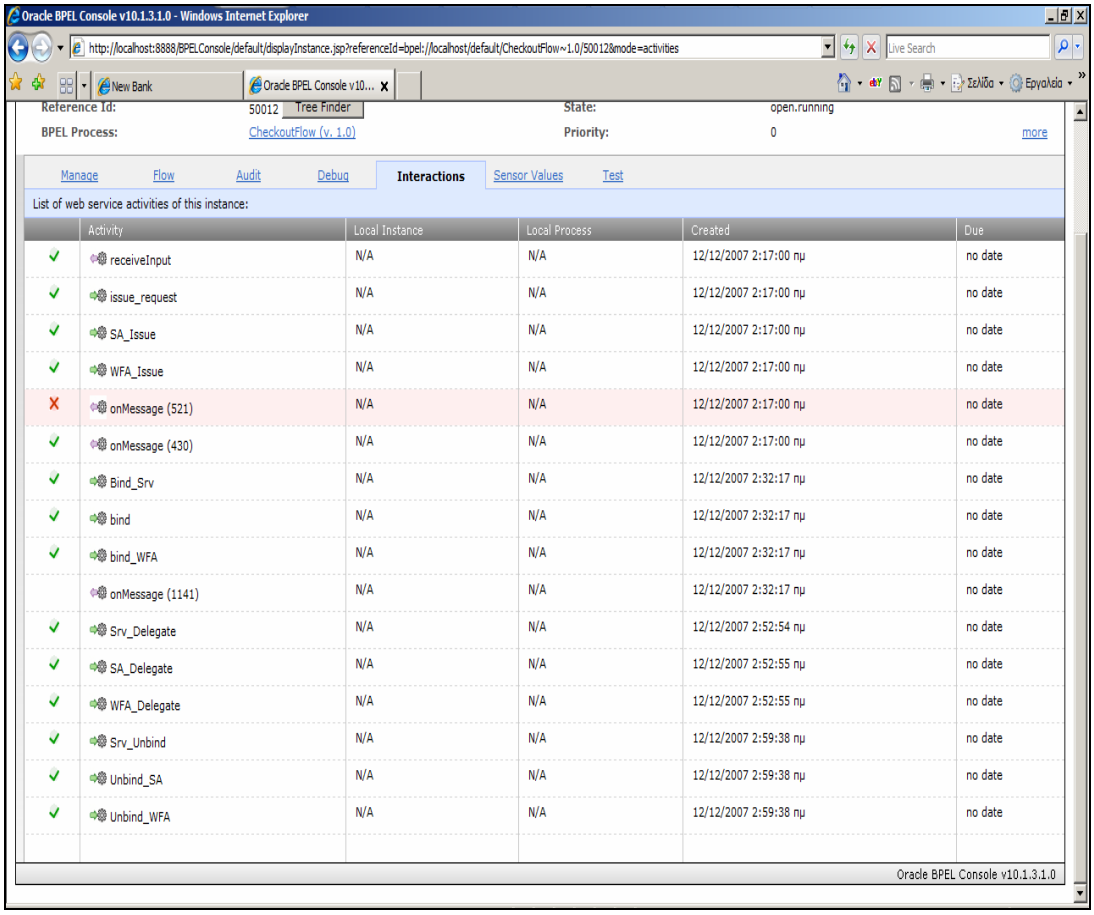


Figure 51. Unbind_Request Event

In Figure 52, the interactions of the client, workflow and services are presented.

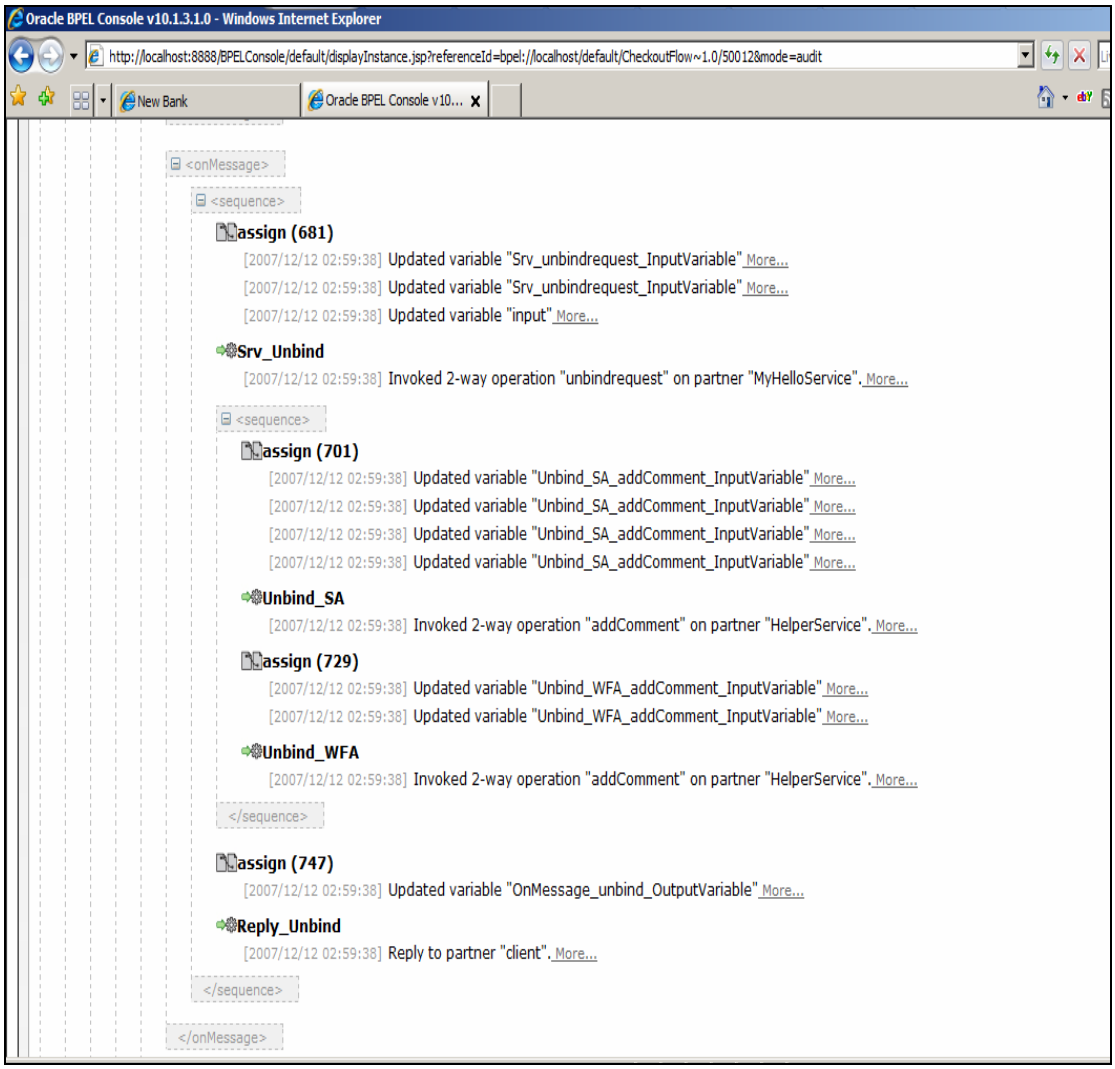
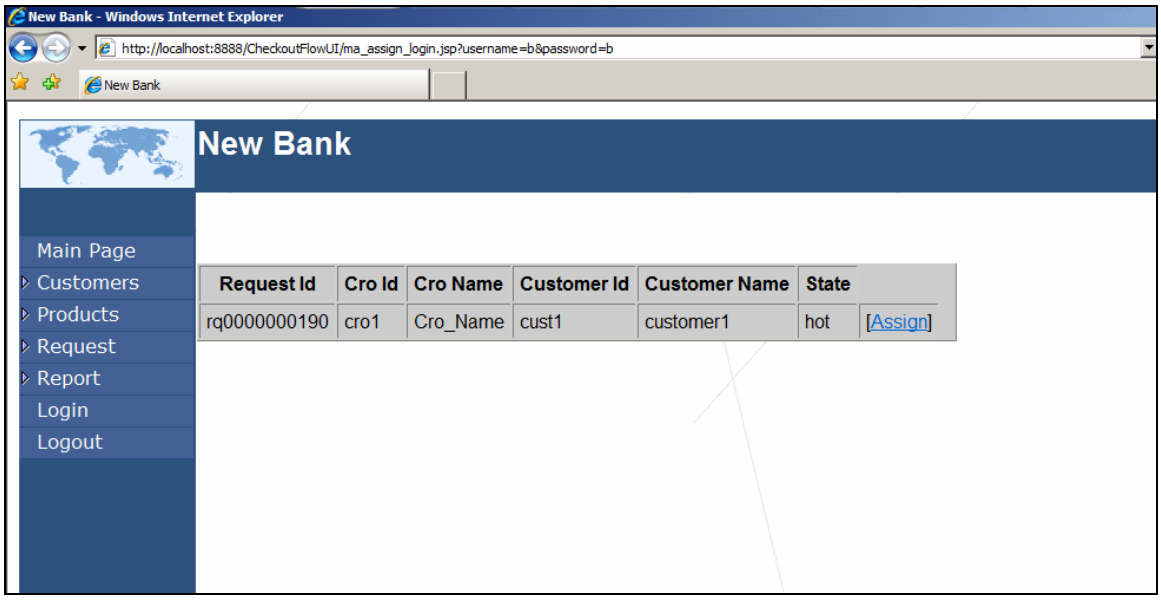


Figure 52. Unbind_Request Event Interactions

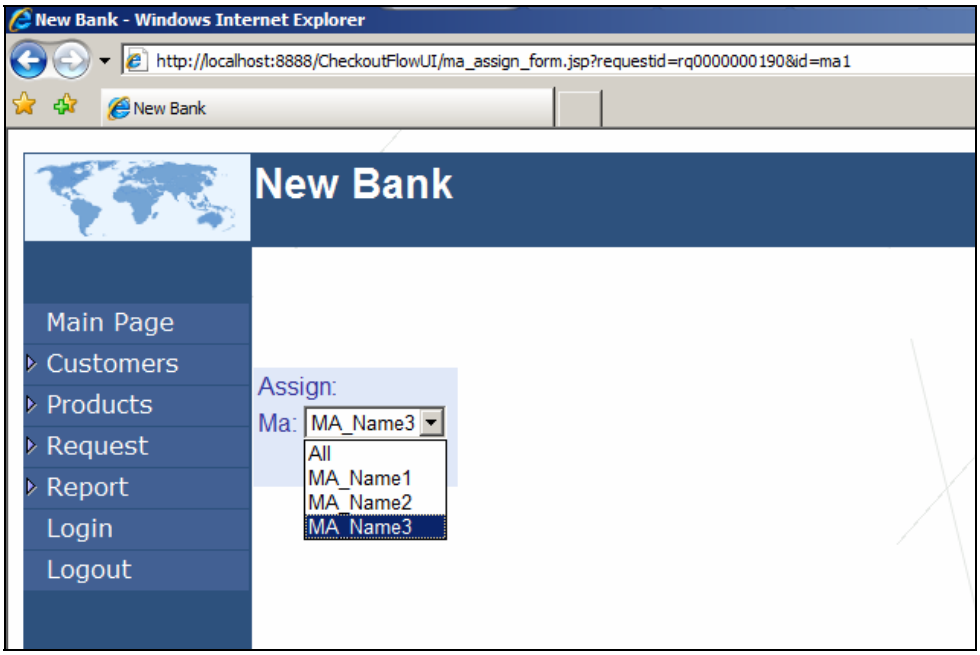
ASSIGN REQUEST

The head of MAs can assign an unbound request, if a certain time has elapsed since the request was issued. When the head of MAs assigns a request, the ConversationId of the respective workflow instance is retrieved from the database and a message is send to that instance. The message describes the workflow operation that will be invoked (the assign operation in this case) and the input parameters.

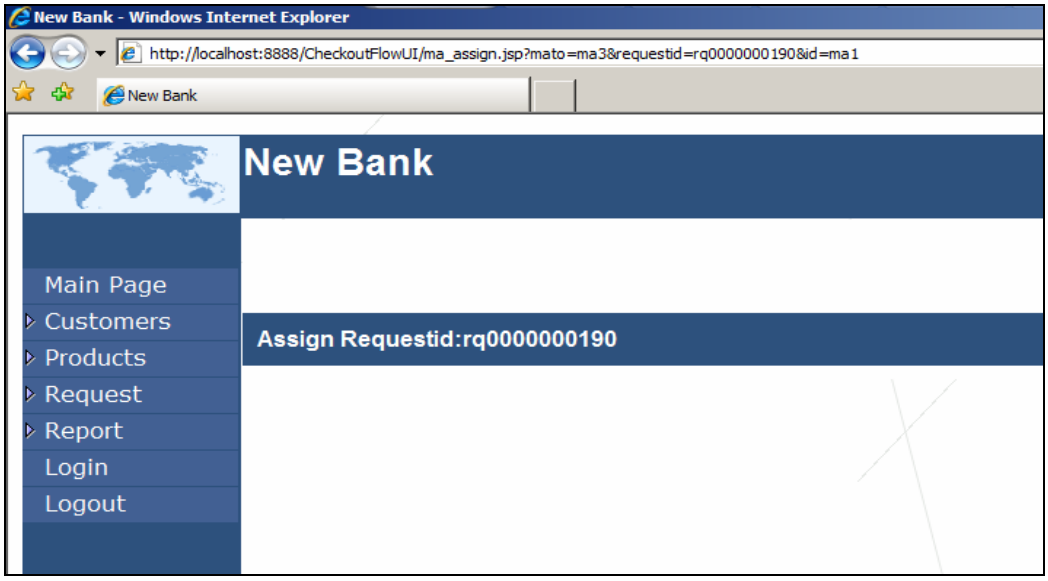
The MA receives a list of the requests that are time critical.



When he selects a request, a list of the originally qualifying MAs is returned.



He assigns the request to one of the listed MAs and the system returns a confirmation message.



First, a web service is invoked by the workflow, to assign the request. Next, the security and workflow agents are invoked, to configure the access rights and the task assignments. Finally, a confirmation message is returned.

In Figure 53, the workflow for the Assign_Request event is presented.

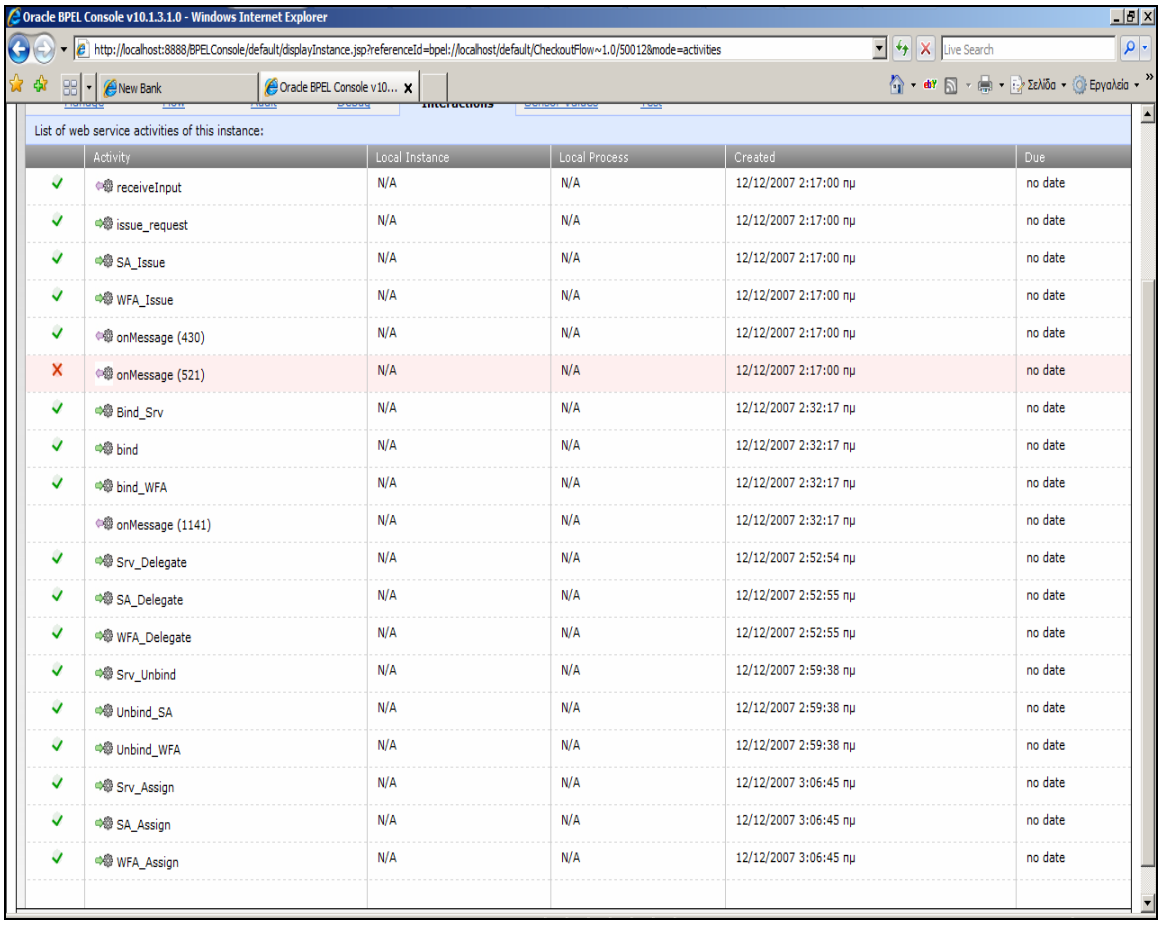


Figure 53. Assign_Request Event

In Figure 54, the interactions of the client, workflow and services are presented.

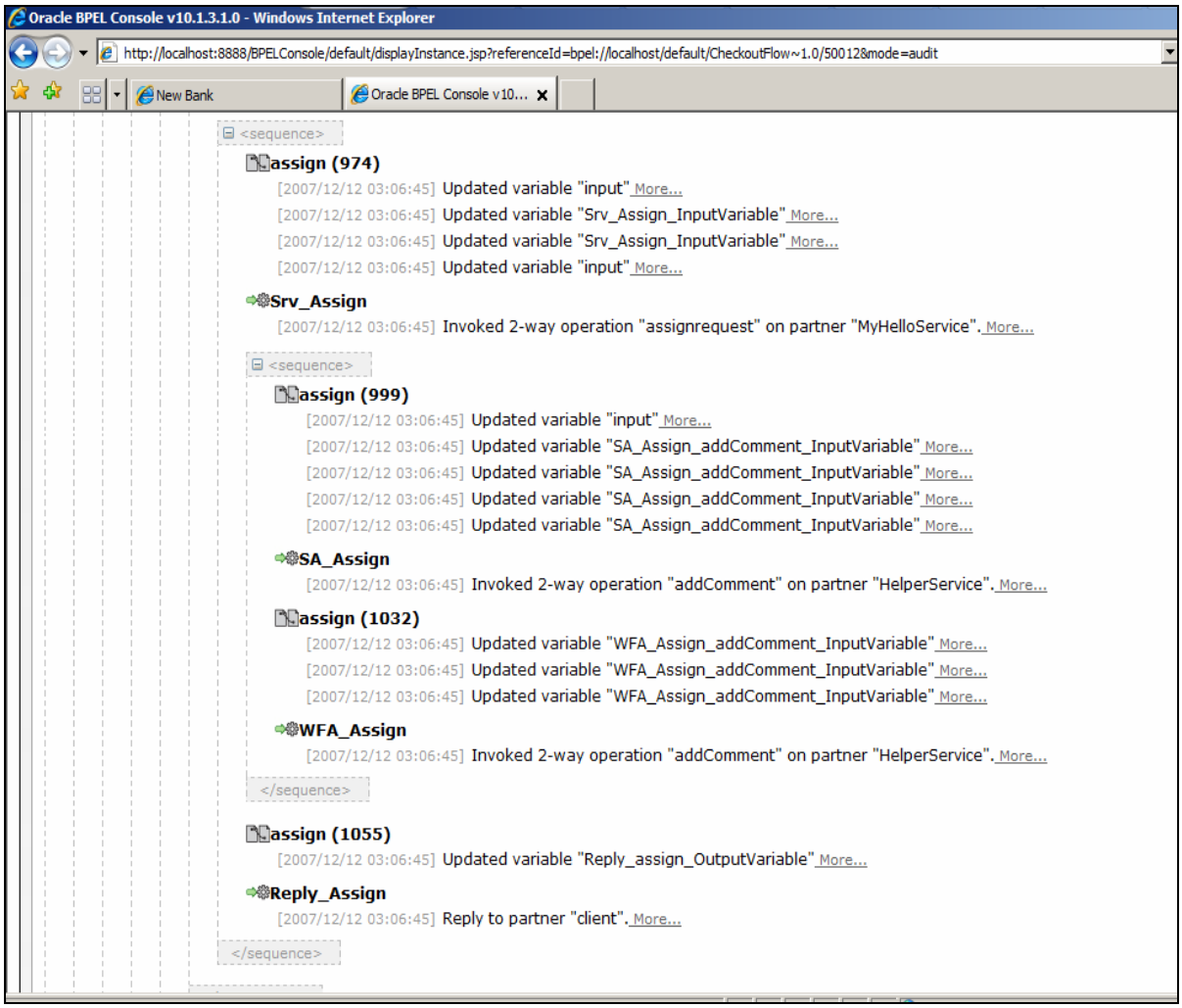
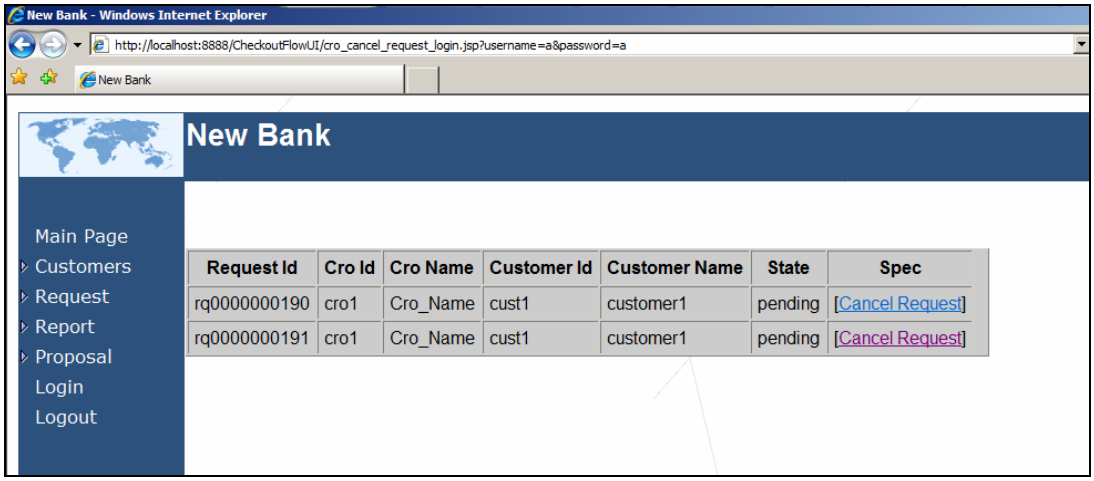


Figure 54. Assign_Request Event Interactions

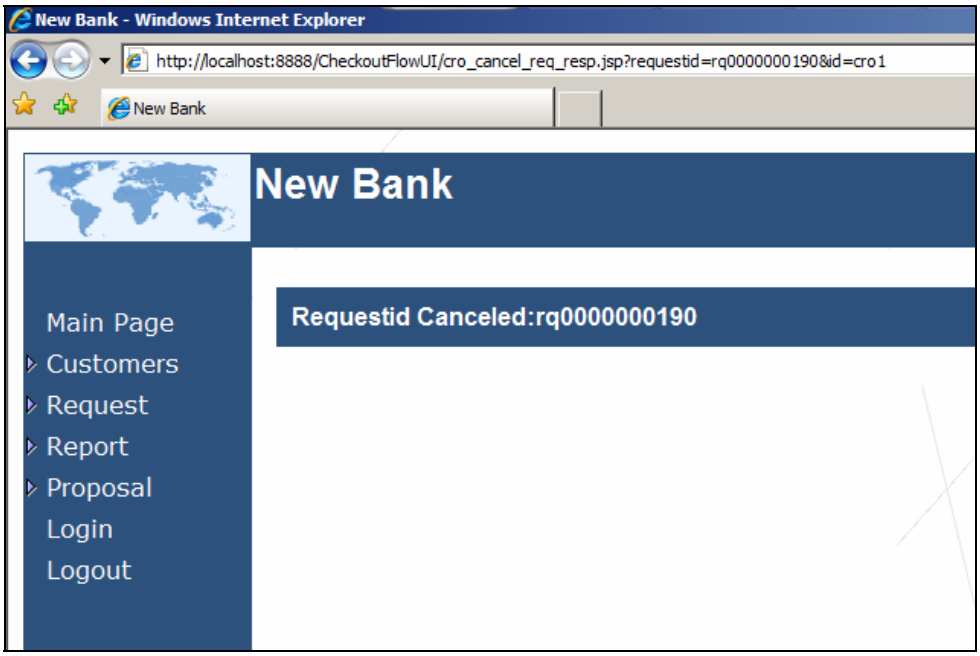
CANCEL REQUEST

Any time after the request was issued, the CRO who issued it can cancel the request. When the CRO cancels a request, the ConversationId of the respective workflow instance is retrieved from the database and a message is send to that instance. The message describes the workflow operation that will be invoked (the cancel operation in this case) and the input parameters. The workflow instance, from the moment it is created, listens for the Cancel_Request event.

The CRO receives a list of the requests he has issued.



The CRO cancels the request and the system returns a confirmation message.



The security and workflow agents are invoked, to remove the access rights and the task assignments and to cancel the request. Next, a confirmation message is returned to the CRO and the workflow instance is terminated.

In Figure 55, the workflow for the Cancel_Request event is presented.

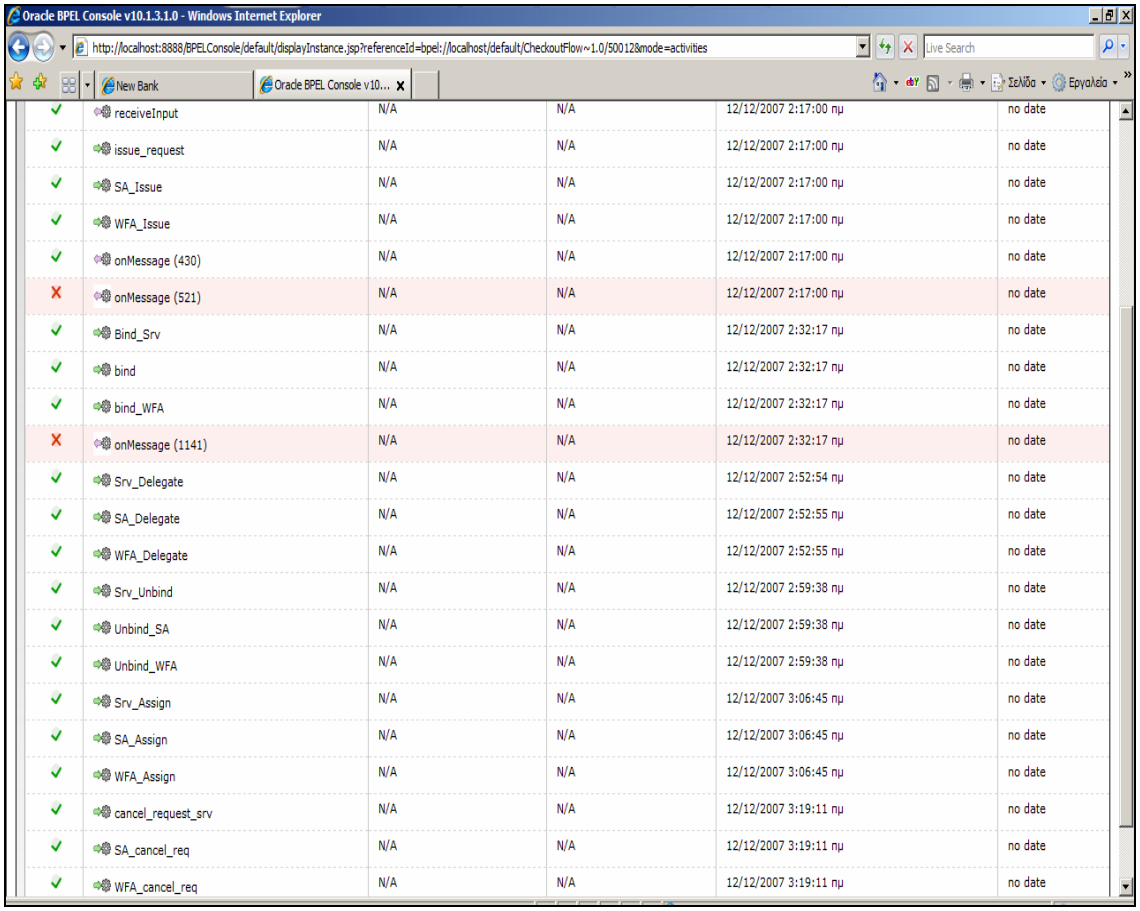


Figure 55. Cancel_Request Event

In Figure 56, the interactions of the client, workflow and services are presented.

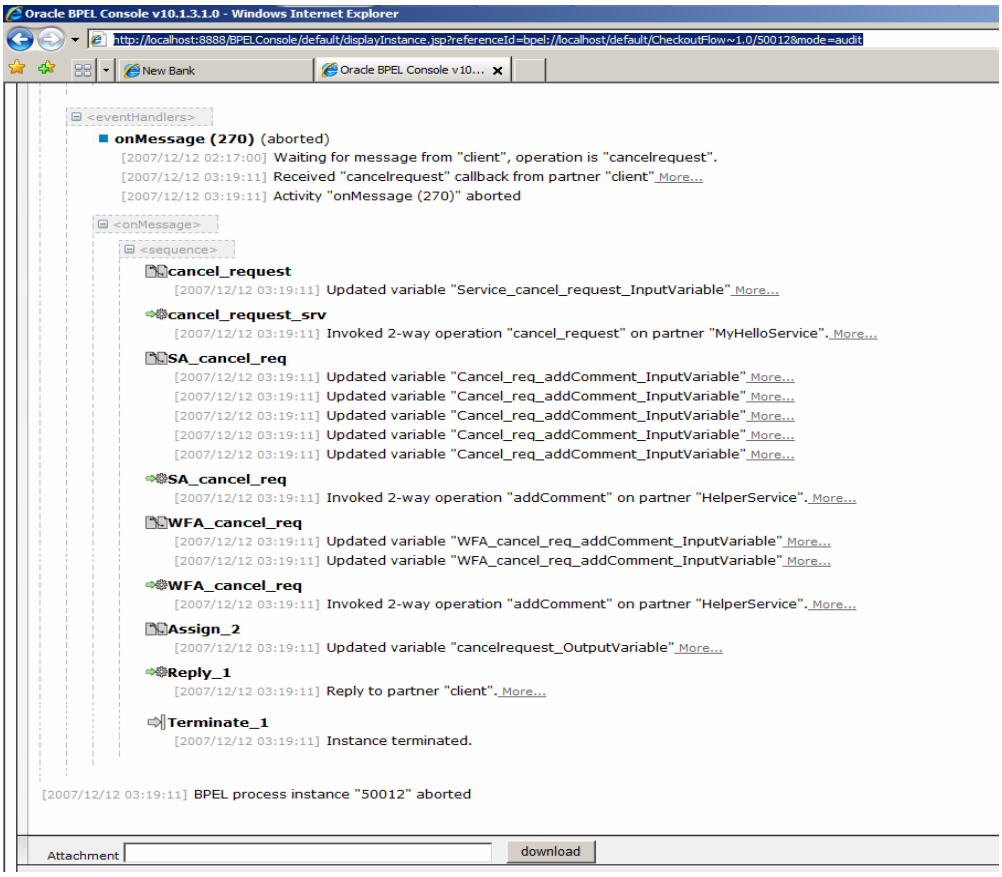


Figure 56. Cancel_Request Event Interactions

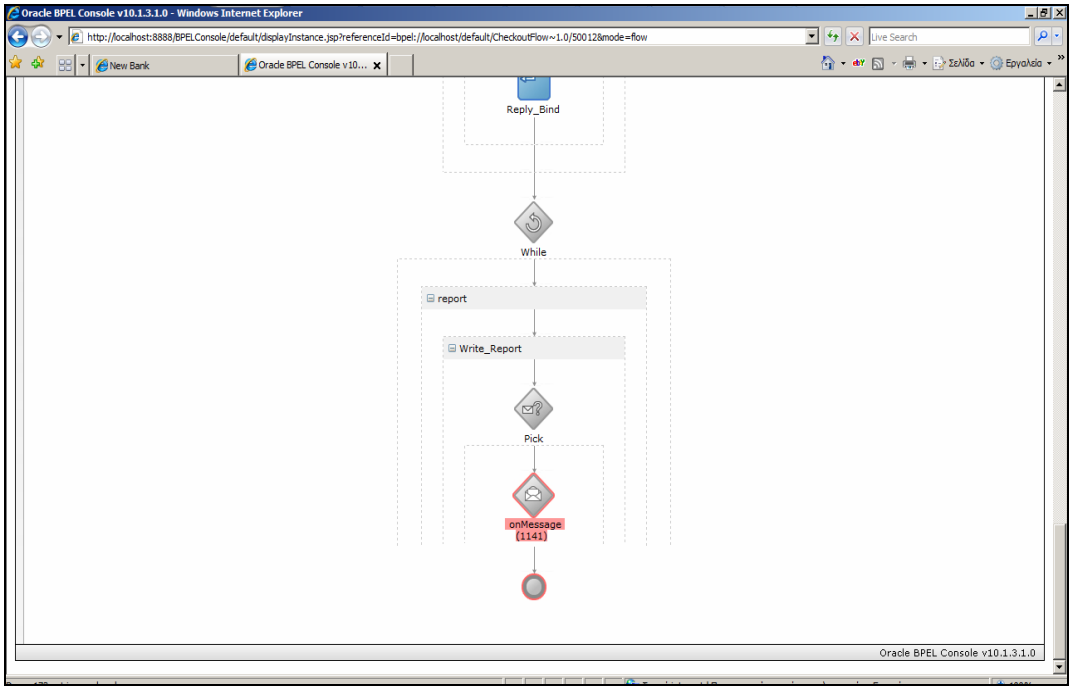
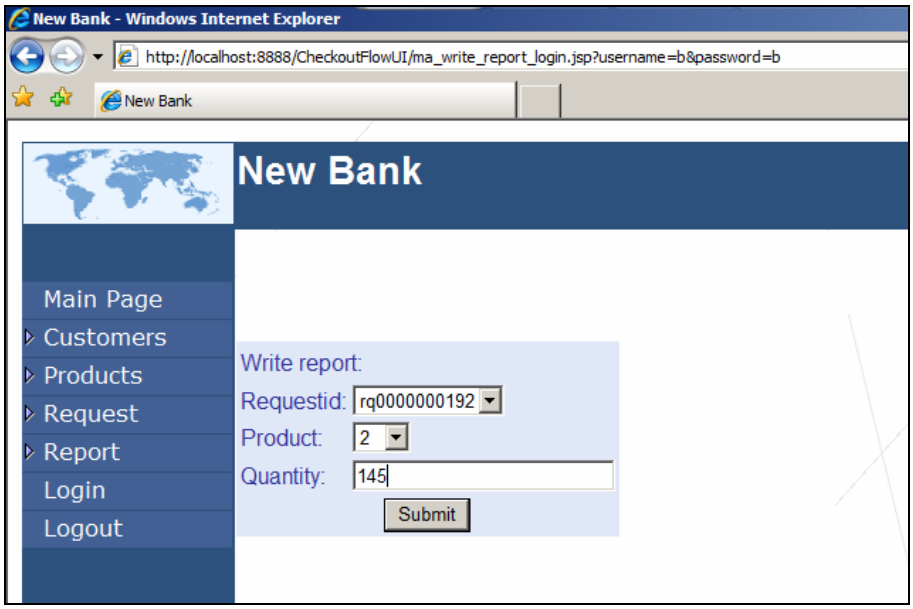


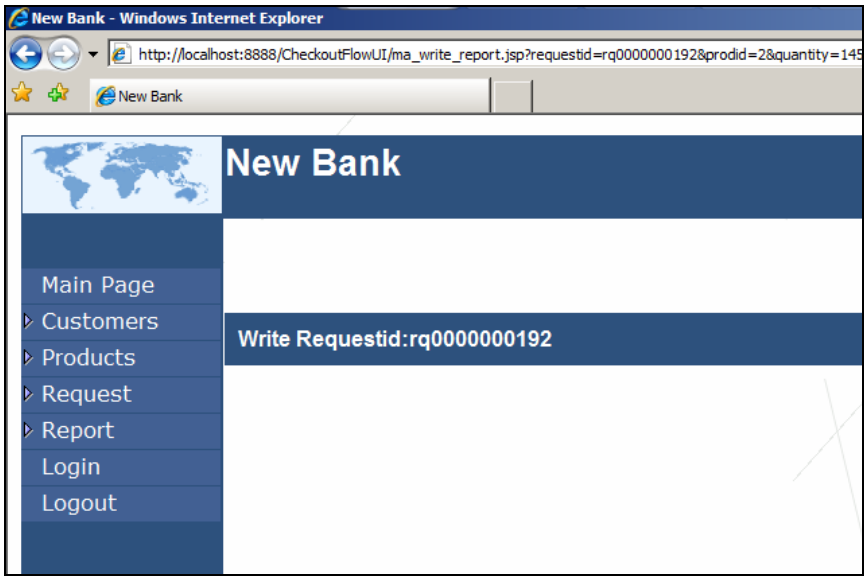
Figure 57. The workflow instance is terminated

WRITE REPORT

A MA writes a report for a request he has bound.



The system returns a confirmation message.



When a MA creates a new report, the ConversationId of the respective workflow instance is retrieved from the database and a message is send to that instance. The message describes the workflow operation that will be invoked (the write_report operation in this case) and the input parameters.

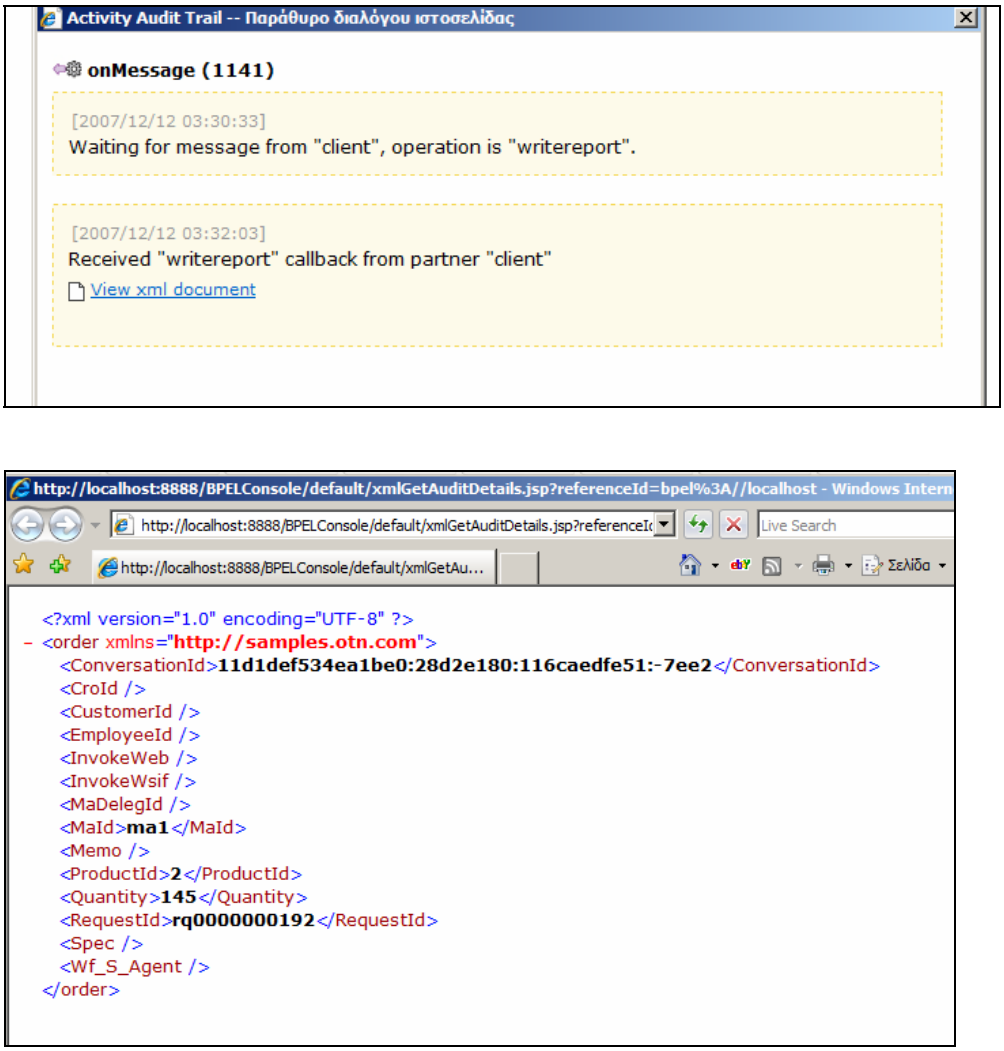


Figure 58. Receive Client Write_Report Message

First, a web service is invoked by the workflow, to create the report. The messages exchanged are the following:

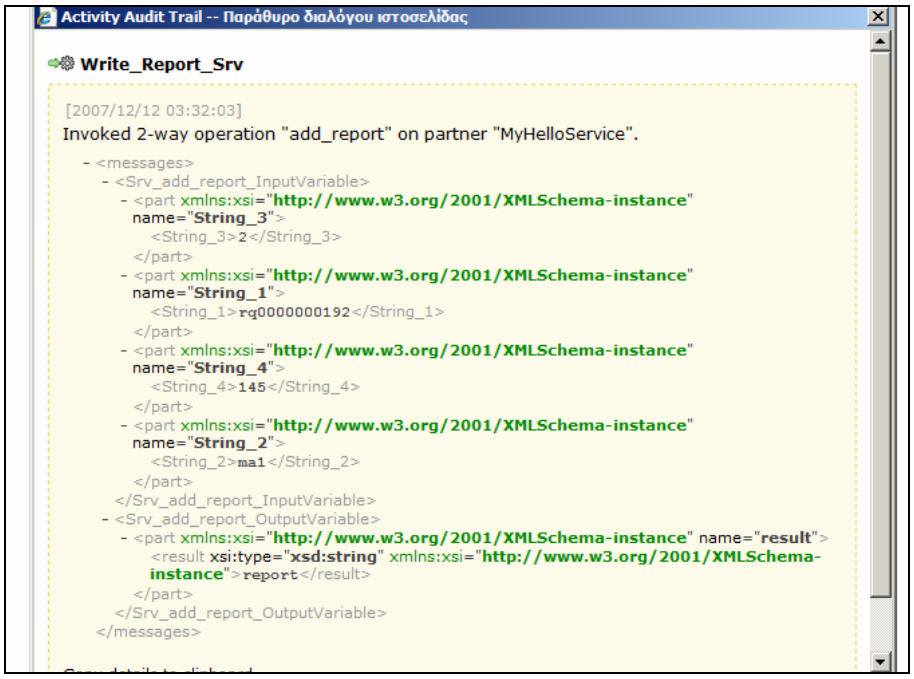


Figure 59. Write_Report Message

Next, the security agent is invoked, to configure the access rights. The messages exchanged are the following:

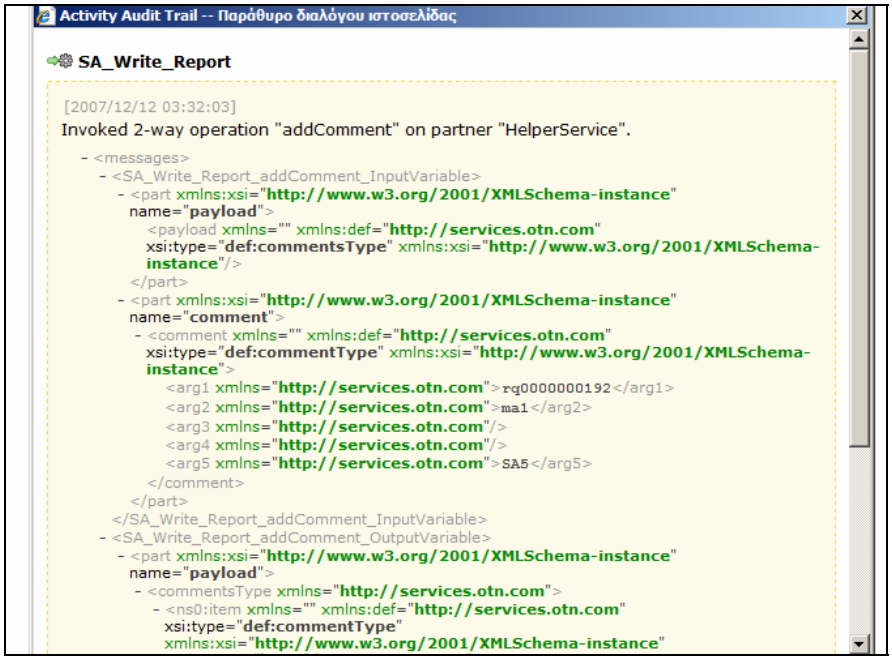


Figure 60. Security Agent Message

A confirmation message is returned to the MA who wrote the report.

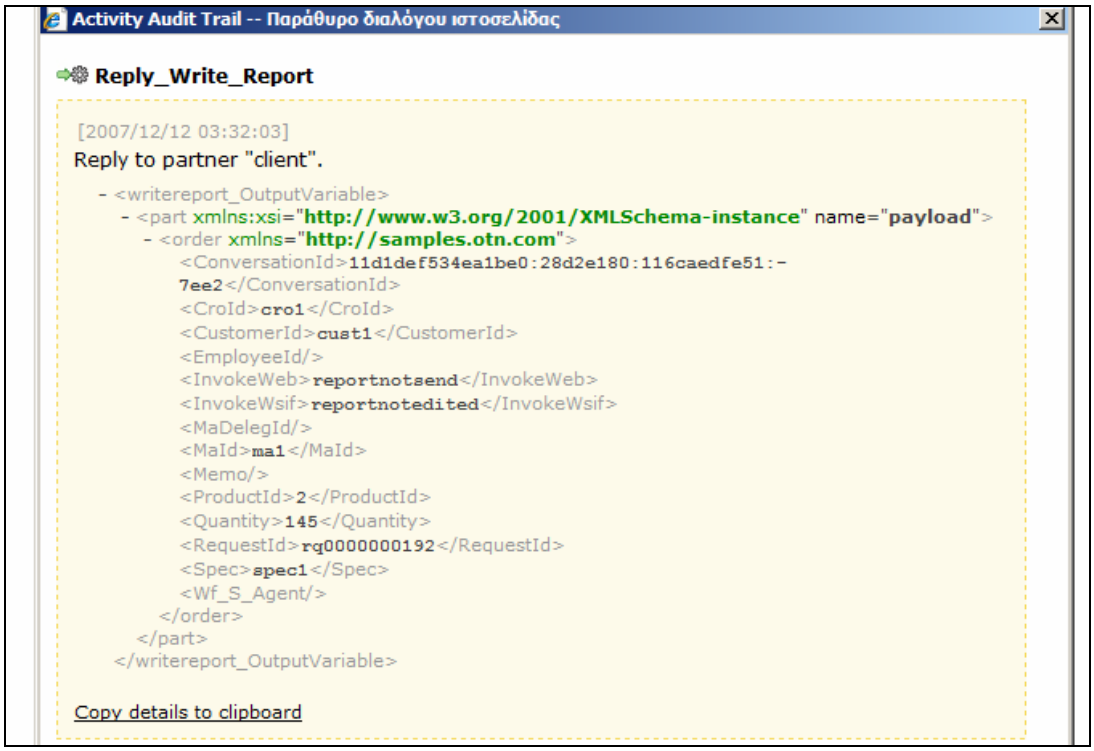


Figure 61. Callback Message

In Figure 62, the interactions of the client, workflow and services are presented.

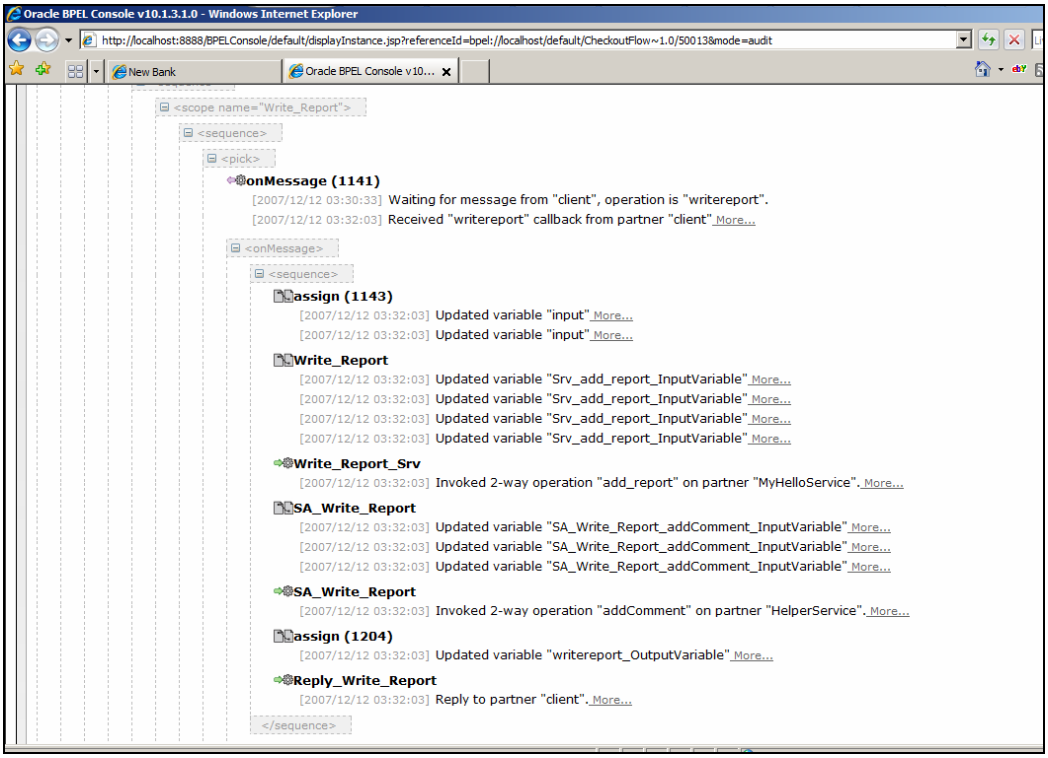


Figure 62. Write_Report Scope Interactions

In Figure 63, the workflow for the Write_Report scope is presented.

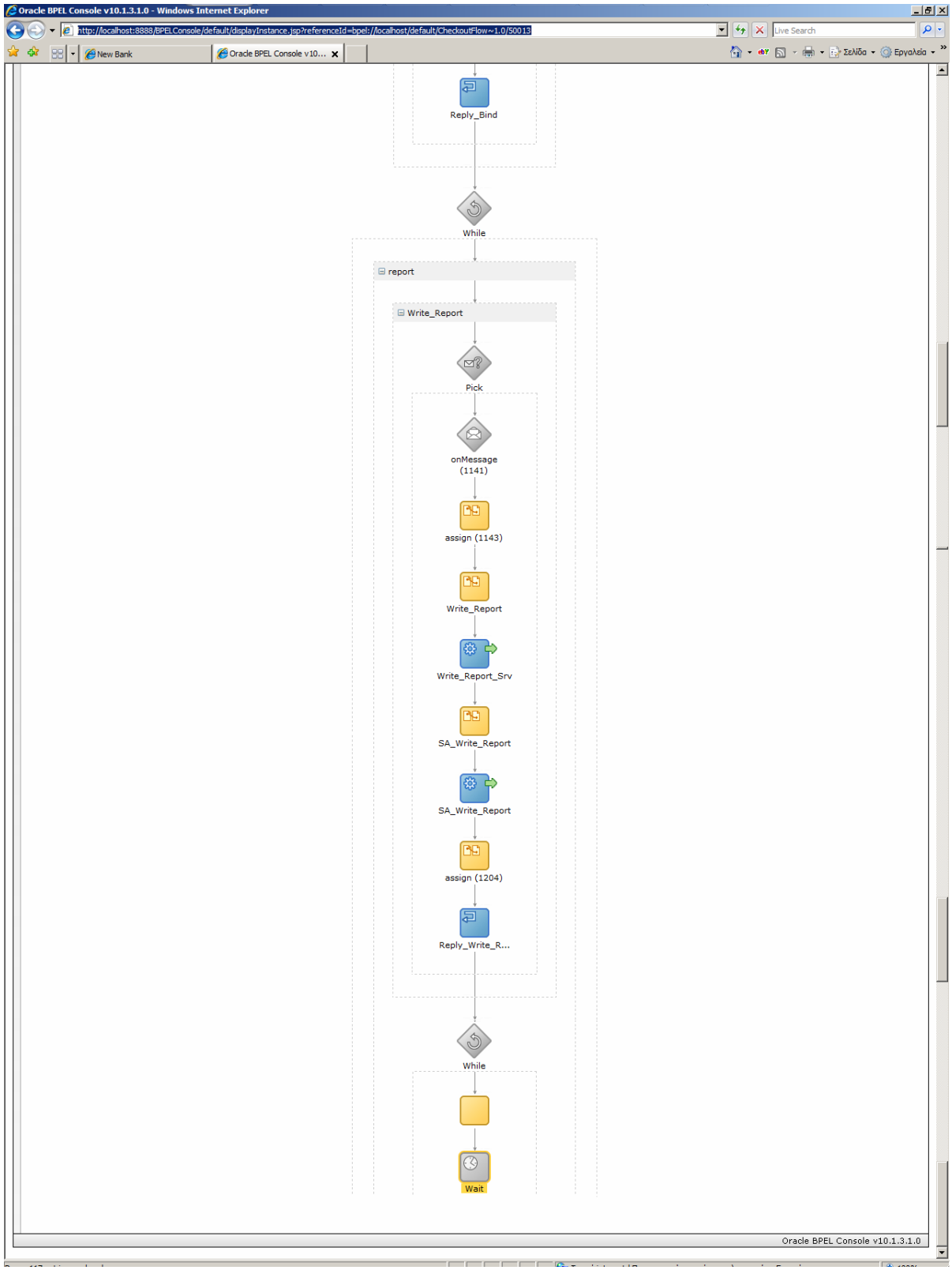
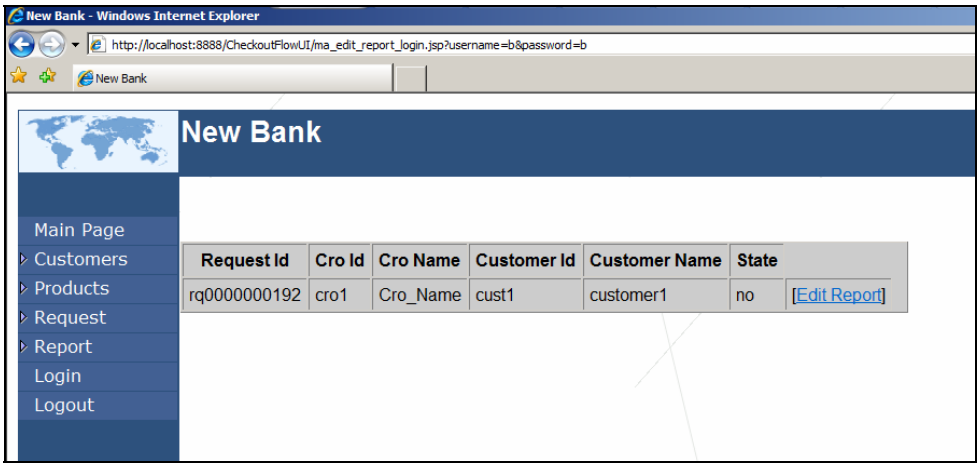


Figure 63. Write_Report Scope

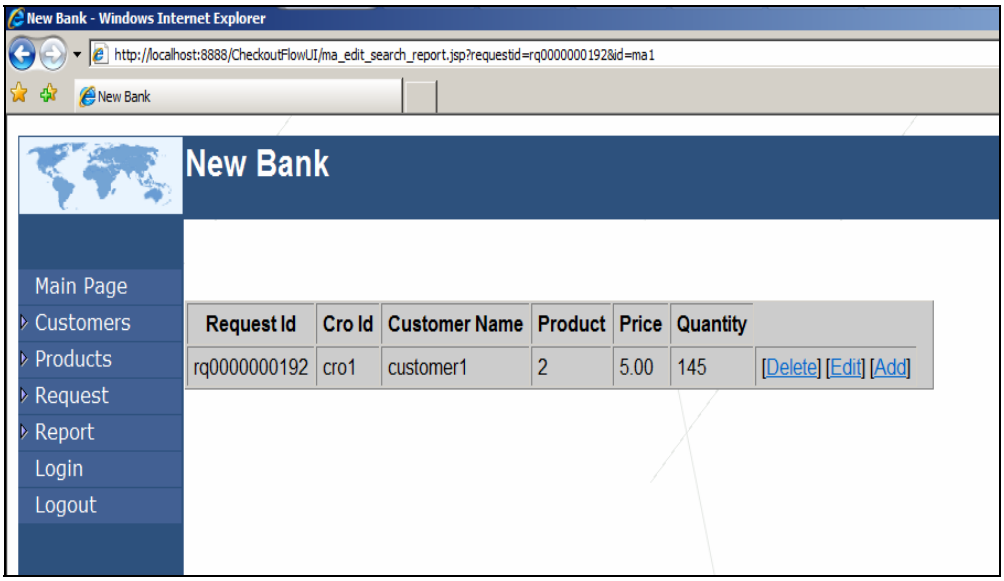
EDIT REPORT

A MA who wrote a report can edit it.

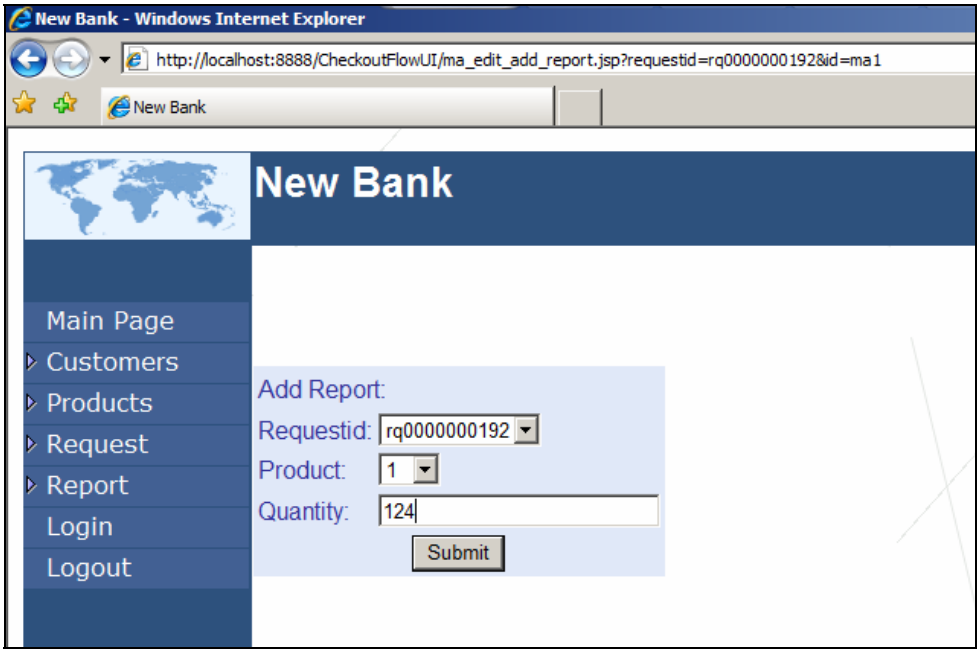
He receives a list of the requests he wrote a report for.



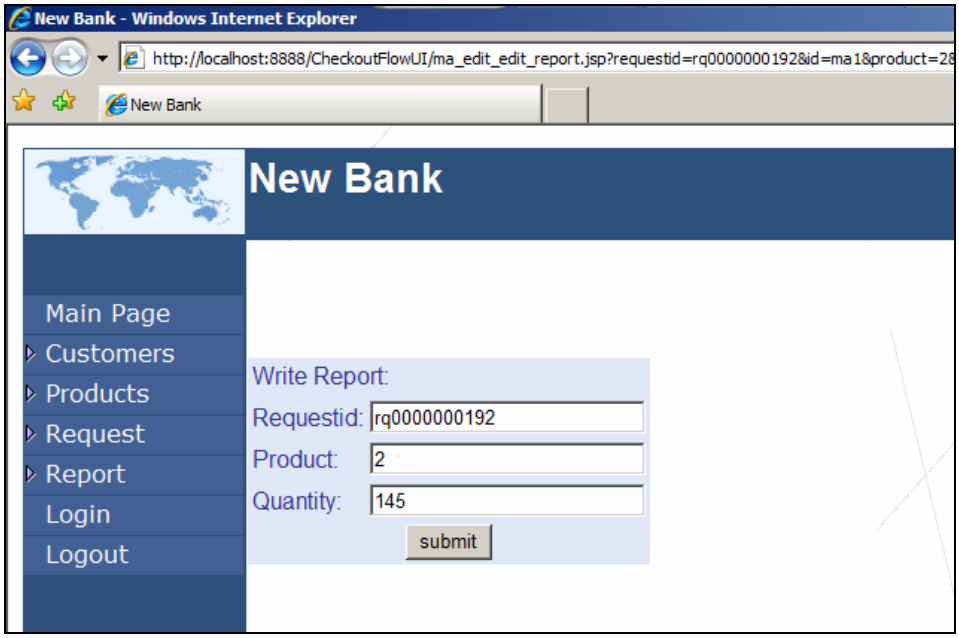
When he selects a request, he receives the report records.



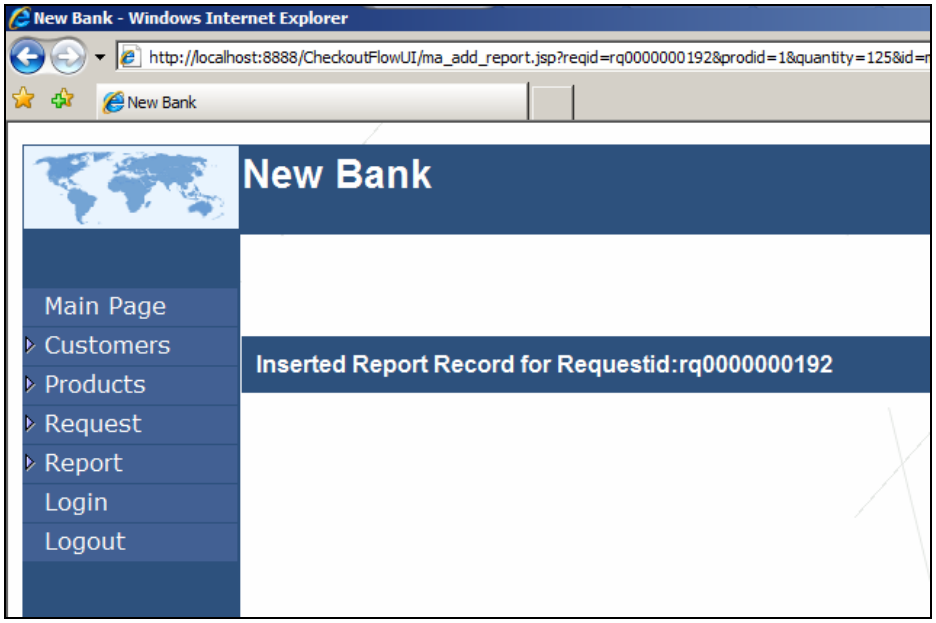
The MA can add new records.



The MA can edit or delete existing records.



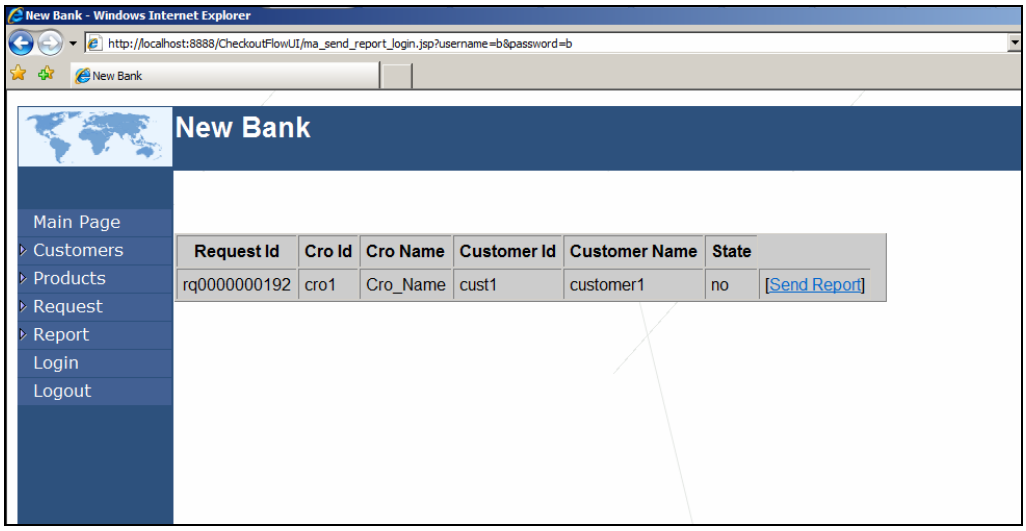
Each time, he receives a confirmation message from the system.



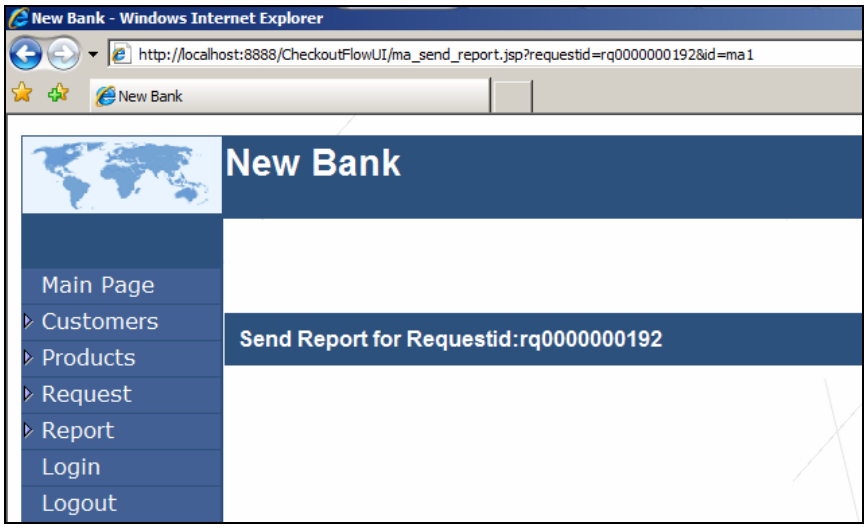
SEND REPORT

A MA sends a report he has written. When a report is send, the ConversationId of the respective workflow instance is retrieved from the database and a message is send to that instance. The message describes the workflow operation that will be invoked (the send_report operation in this case) and the input parameters.

He MA sends the report.



The system returns a confirmation message.



First, a web service is invoked by the workflow, to send the request. Next, the security and workflow agents are invoked, to configure the access rights and the task assignments. Finally, a confirmation message is returned.

In Figure 64, the workflow for the Send_Report event is presented.

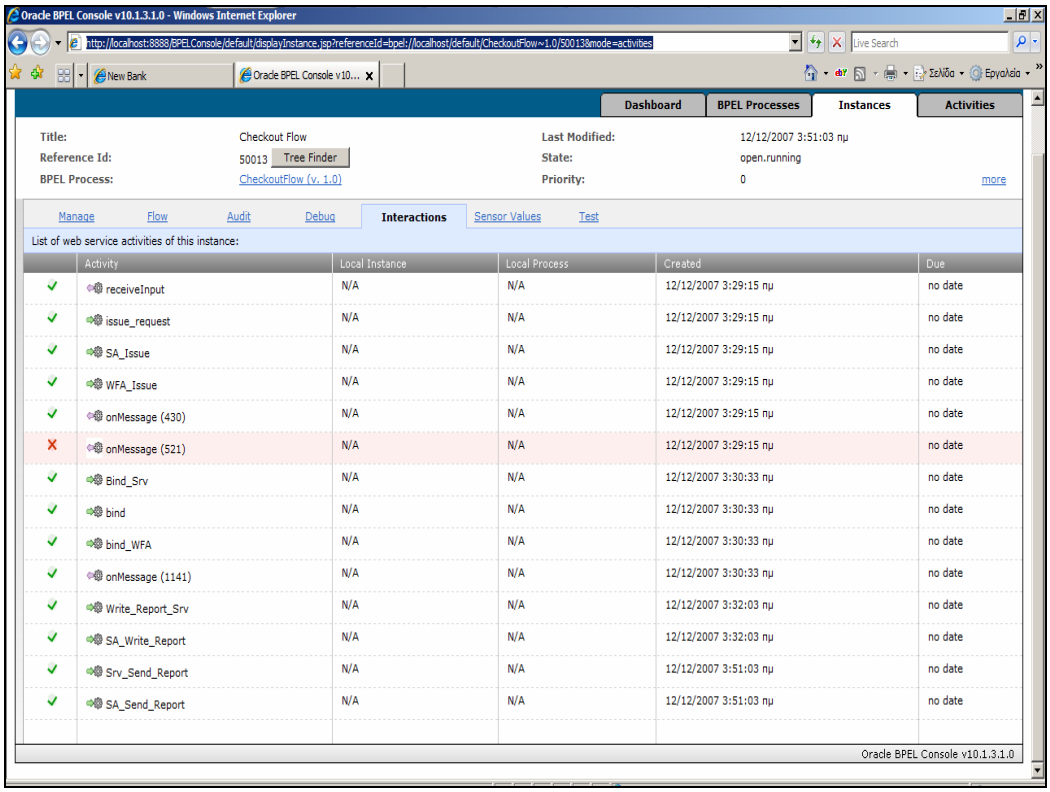


Figure 64. Send_Report Event

In Figure 65, the interactions of the client, workflow and services are presented.

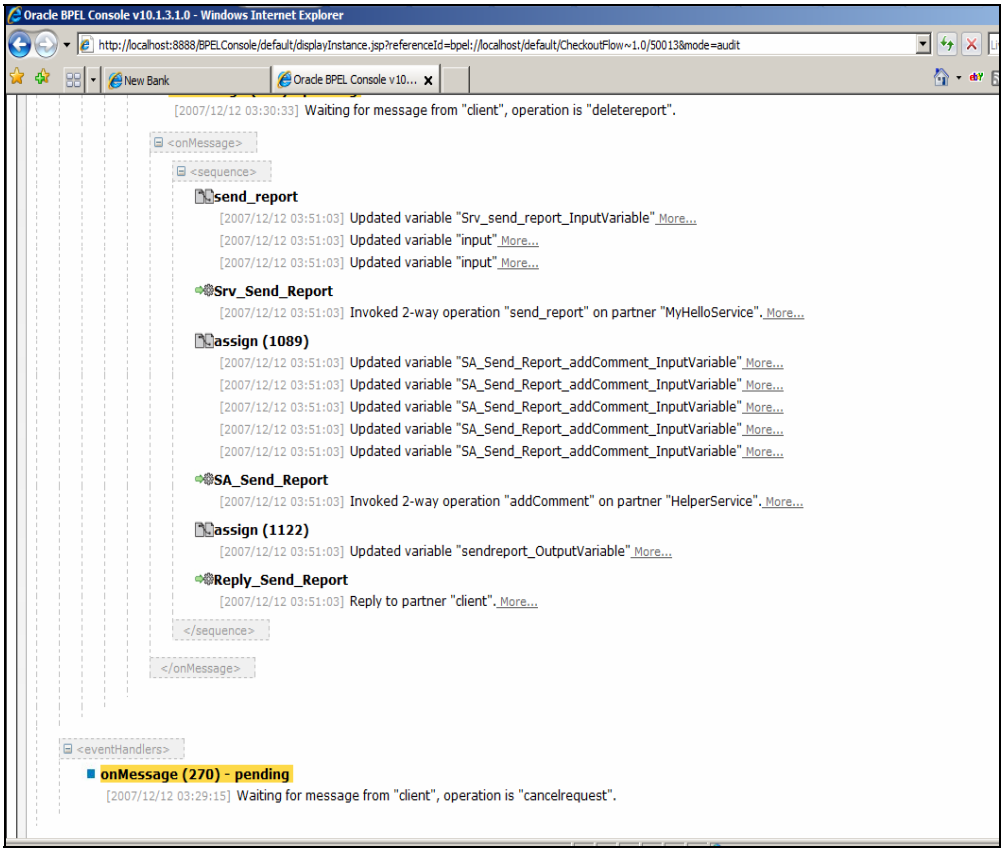
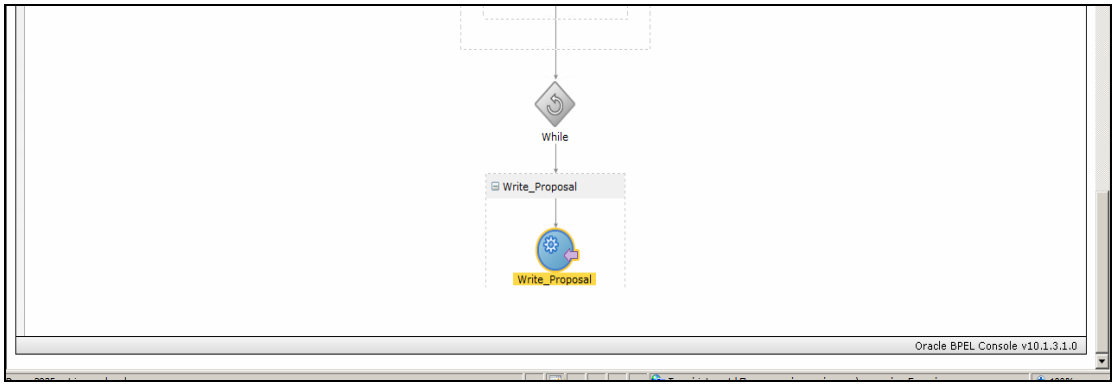


Figure 65. Send_Report Event Interactions

After the Report scope, the workflow blocks waiting for the CRO to write a proposal based on the report he has received.



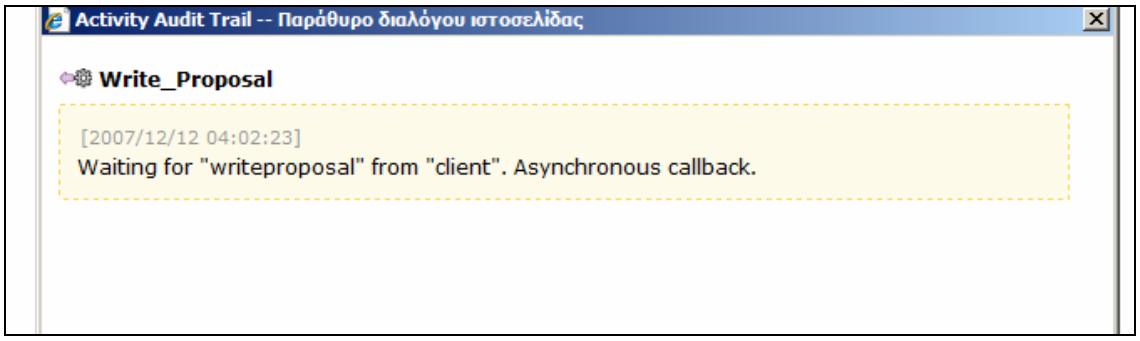
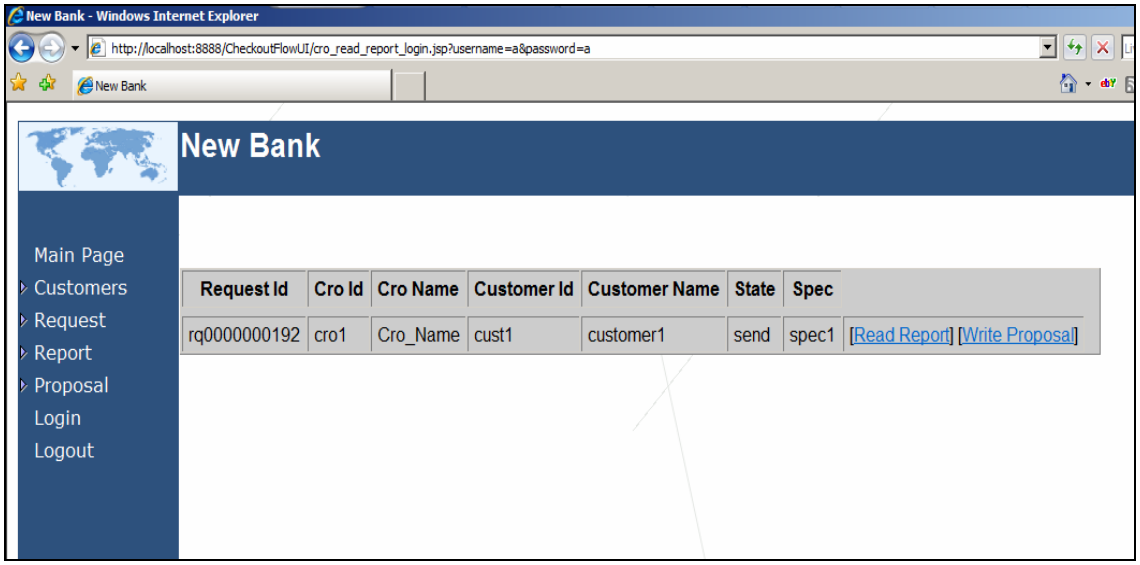


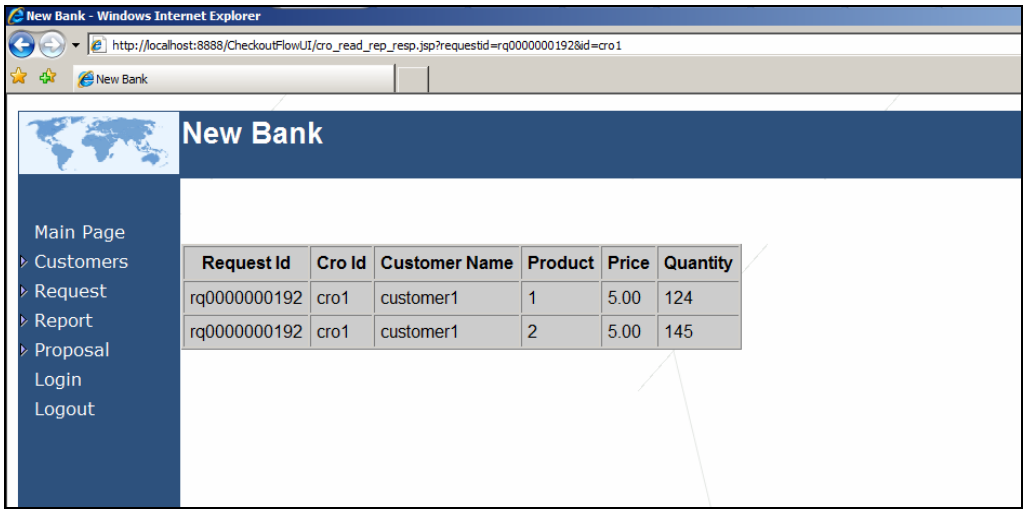
Figure 66. Waiting for Client Message

9.3 WRITE PROPOSAL

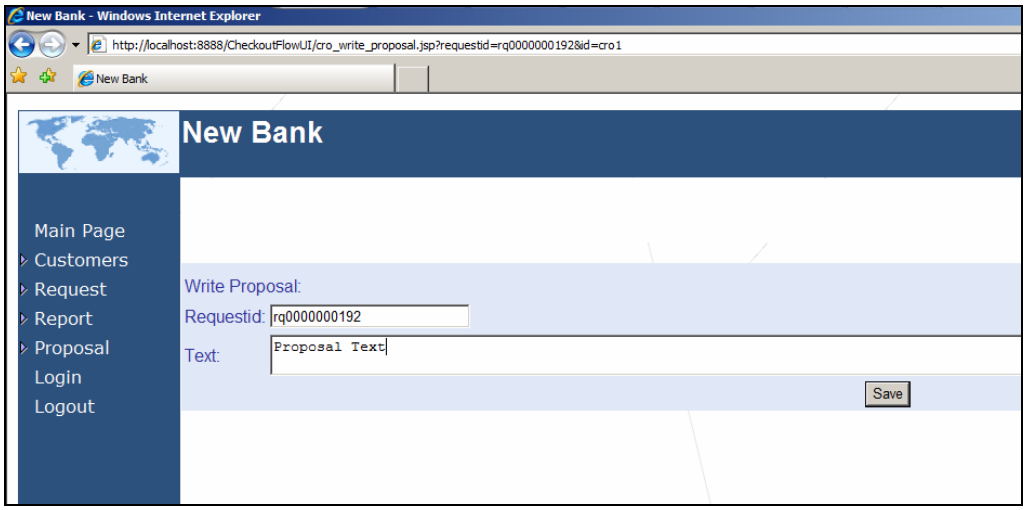
When a CRO searches for new reports assigned to him, a list of requests he has issued, for which reports have been send, is returned.



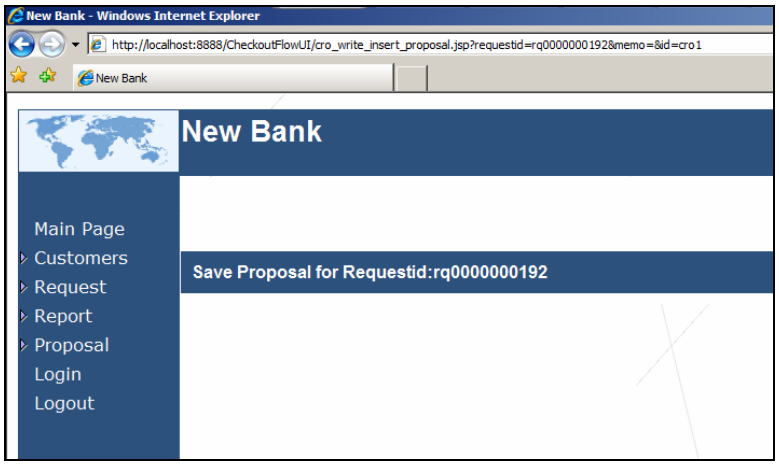
The CRO can select a request and read the respective report.



The CRO can white a proposal based on that report.



When the CRO saves the proposal, he receives a confirmation message from the system.



When a CRO writes a proposal, the ConversationId of the respective workflow instance is retrieved from the database and a message is send to that instance. The message describes the workflow operation that will be invoked (the write_proposal operation in this case) and the input parameters.

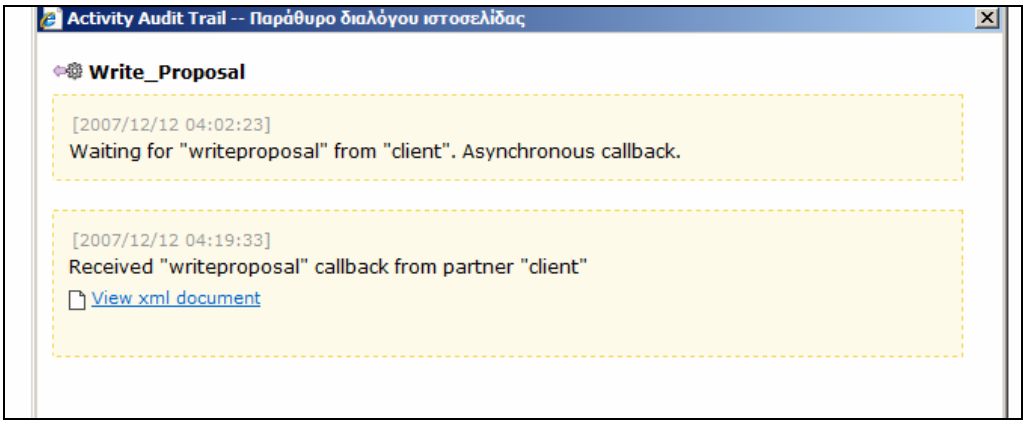


Figure 67. Receive Client Write_Proposal Message

First, a web service is invoked by the workflow, to insert the proposal in the database. The messages exchanged are the following:

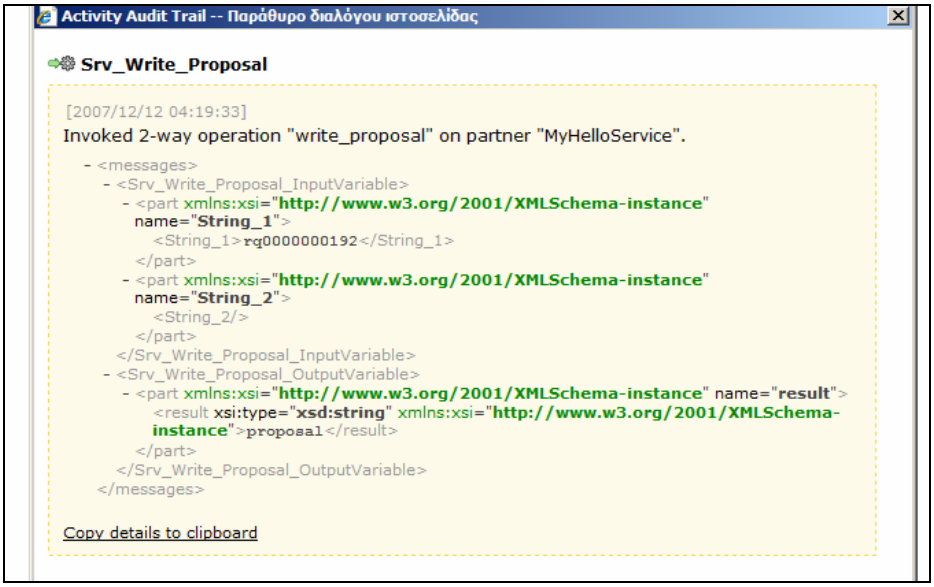


Figure 68. Write_Proposal Message

Next, the security agent is invoked, to configure the access rights and the task assignments. The messages exchanged are the following

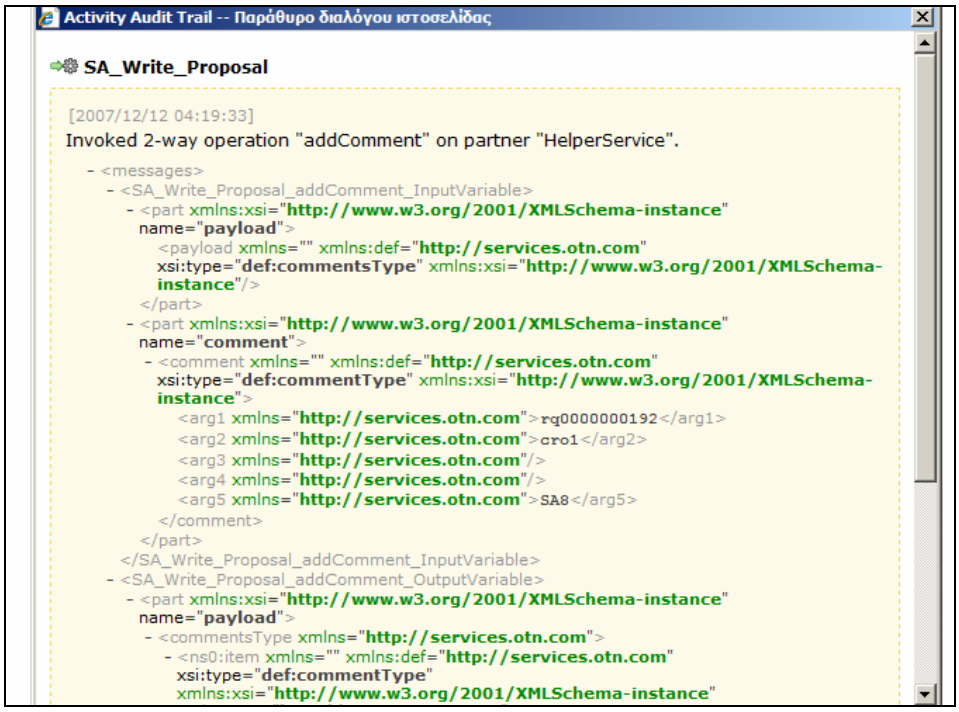


Figure 69. Security Agent Message

A confirmation message is returned to the CRO who issued the request.

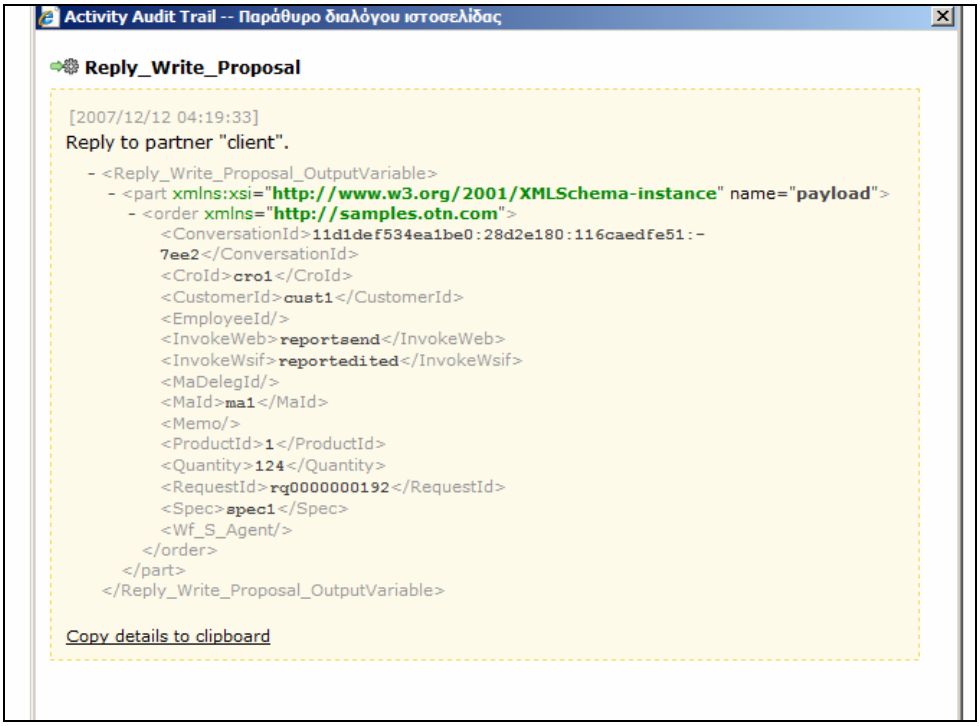


Figure 70. Callback Message

In Figure 71, the interactions of the client, workflow and services are presented.

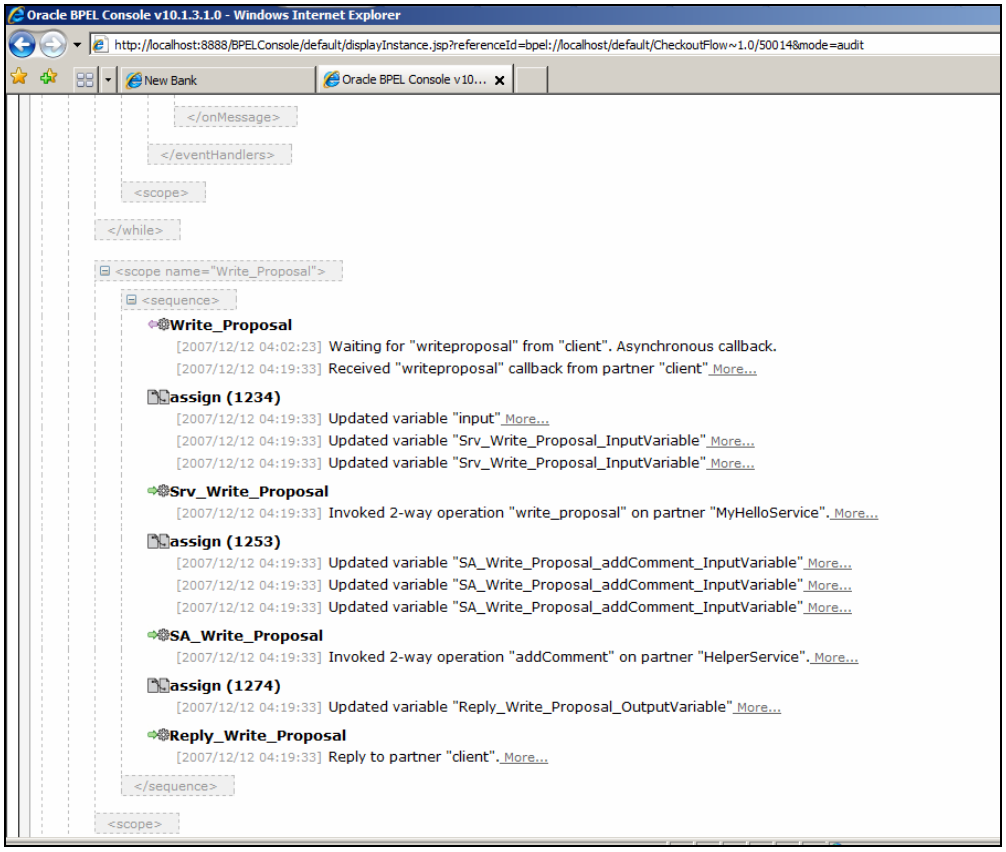


Figure 71. Write_Proposal Scope Interactions

After the Write_Proposal scope, the workflow blocks waiting for the CRO to send the proposal.

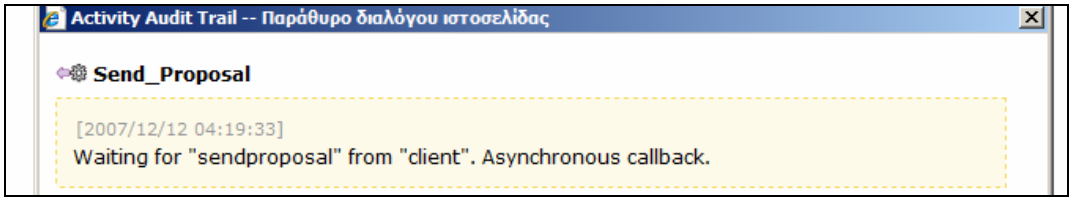


Figure 72. Waiting for Client Message

In Figure 73, the workflow for the Write_Proposal scope is presented.

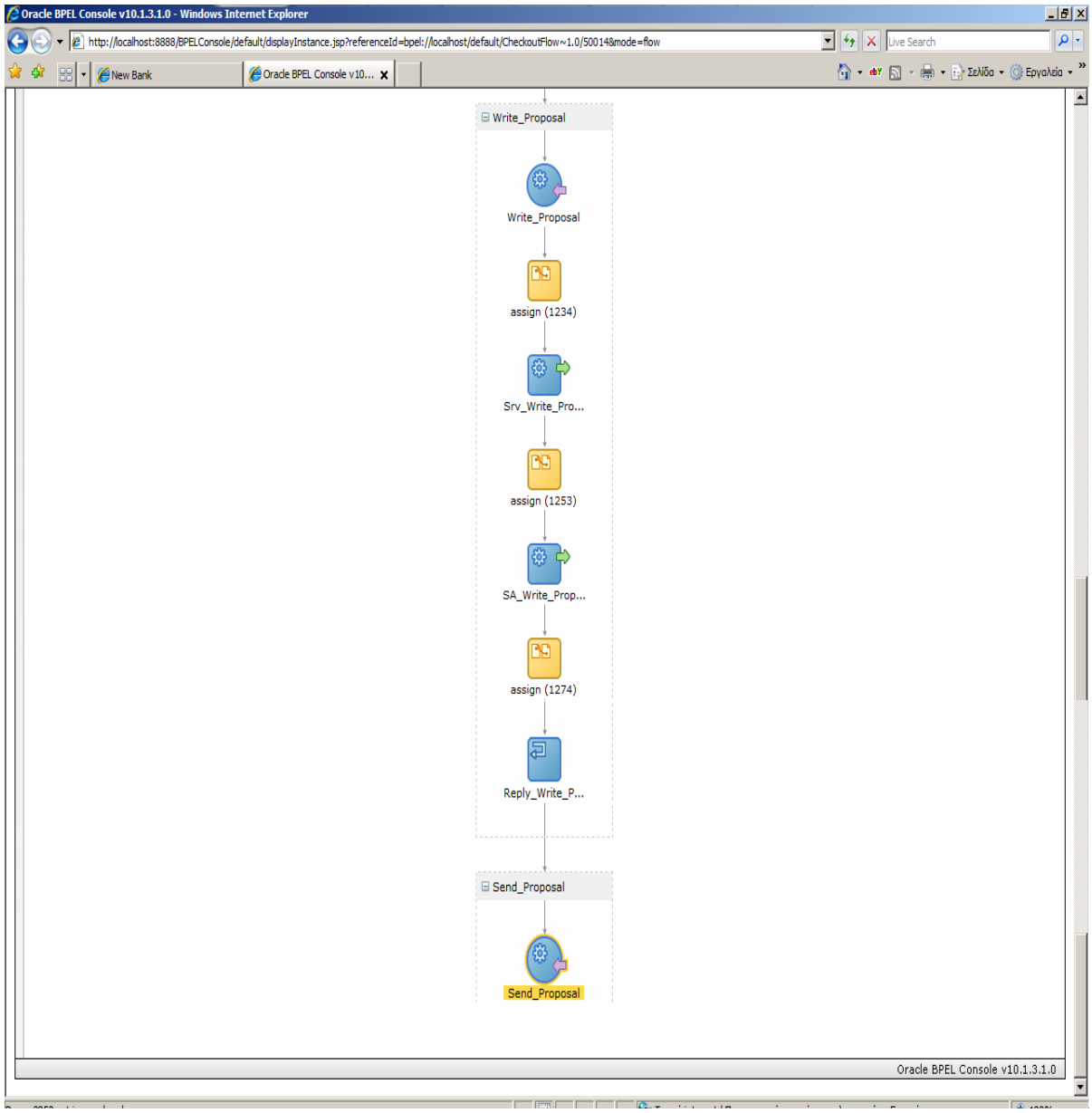
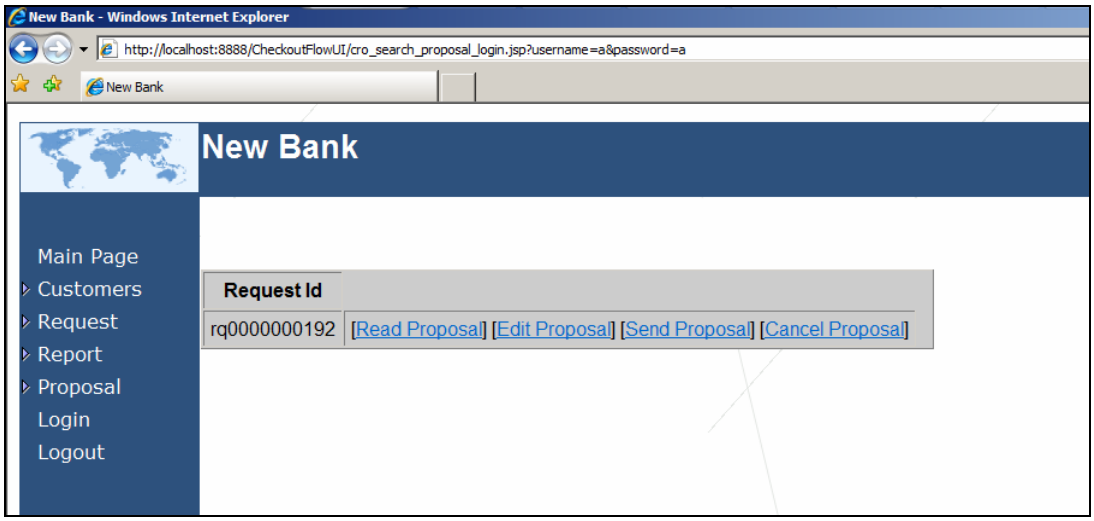


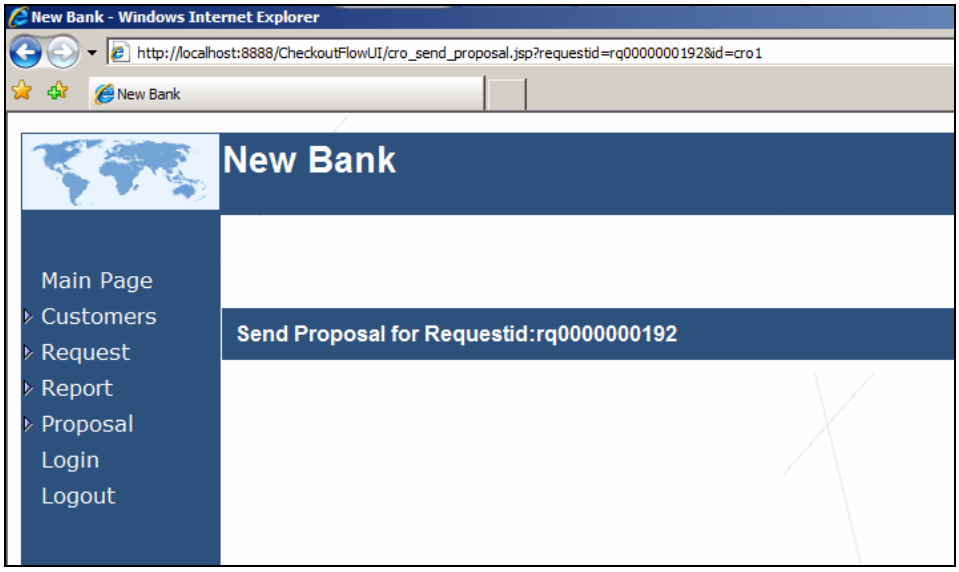
Figure 73. Write_Proposal Scope

EDIT – SEND PROPOSAL

The CRO can read, edit and delete and send a proposal he has written.



When the CRO sends the proposal, he receives a confirmation message from the system.



When a CRO sends a proposal, the ConversationId of the respective workflow instance is retrieved from the database and a message is send to that instance. The message describes the workflow operation that will be invoked (the send_proposal operation in this case) and the input parameters.

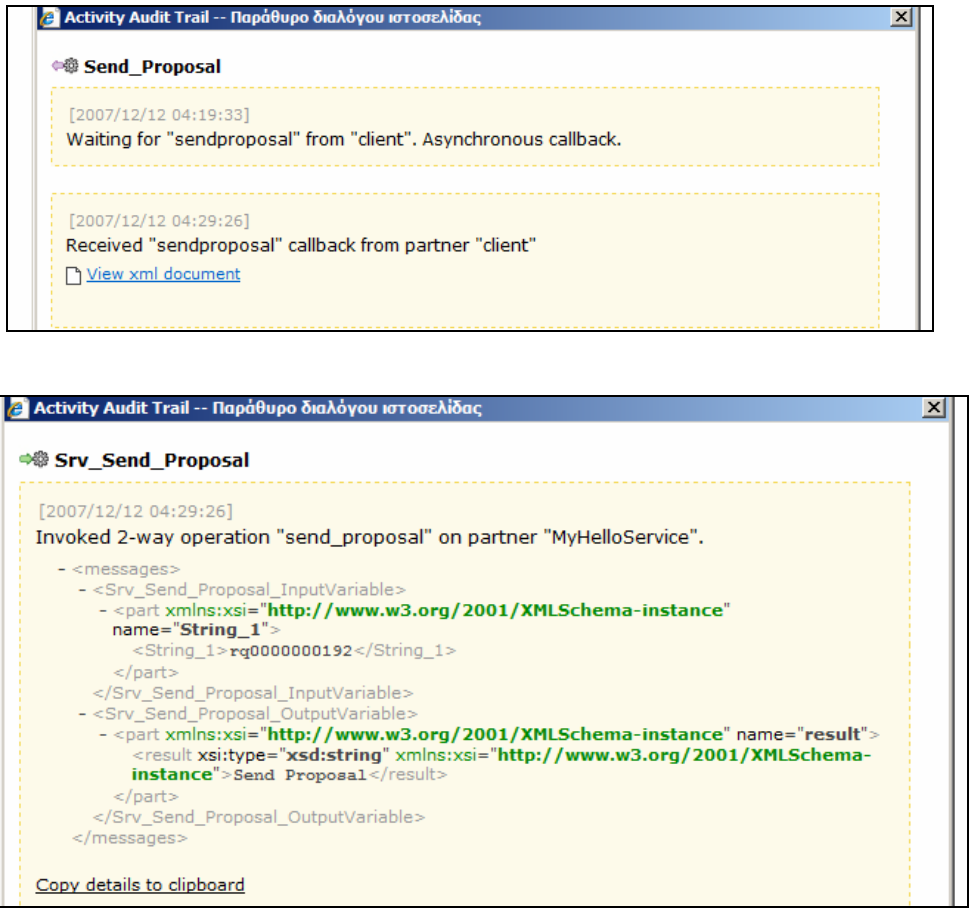


Figure 74. Receive Client Send_Proposal Message

First, a web service is invoked by the workflow, to send the proposal.

Next, the security agent is invoked, to configure the access rights and the task assignments.

The messages exchanged are the following:

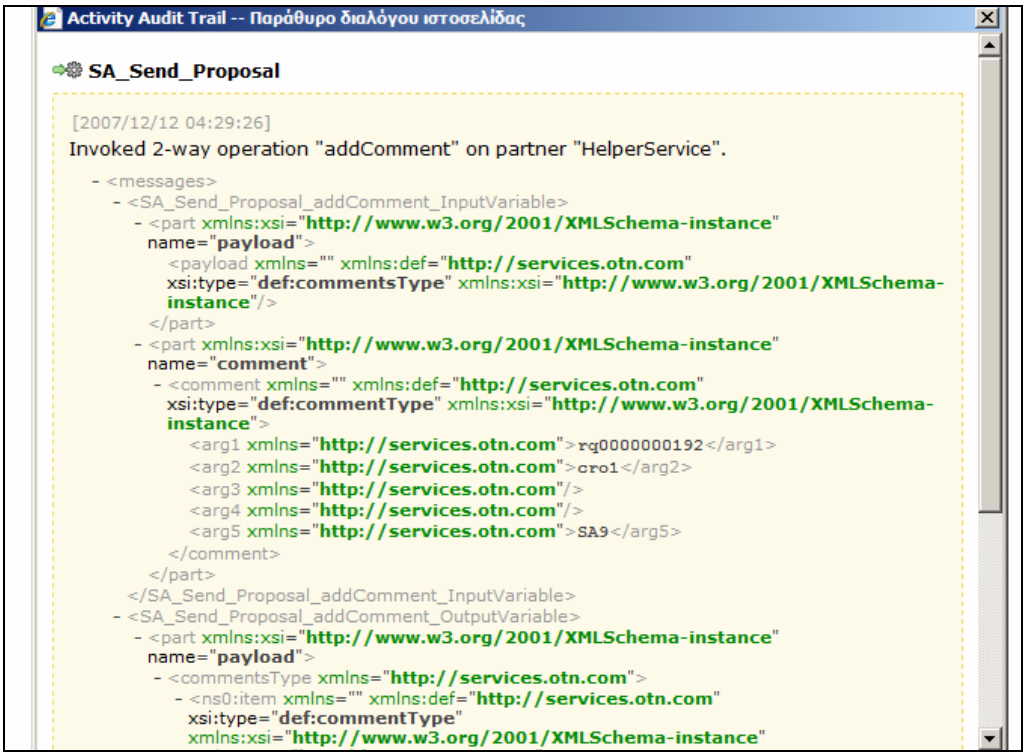


Figure 75. Security Agent Message

A confirmation message is returned to the CRO who issued the request.

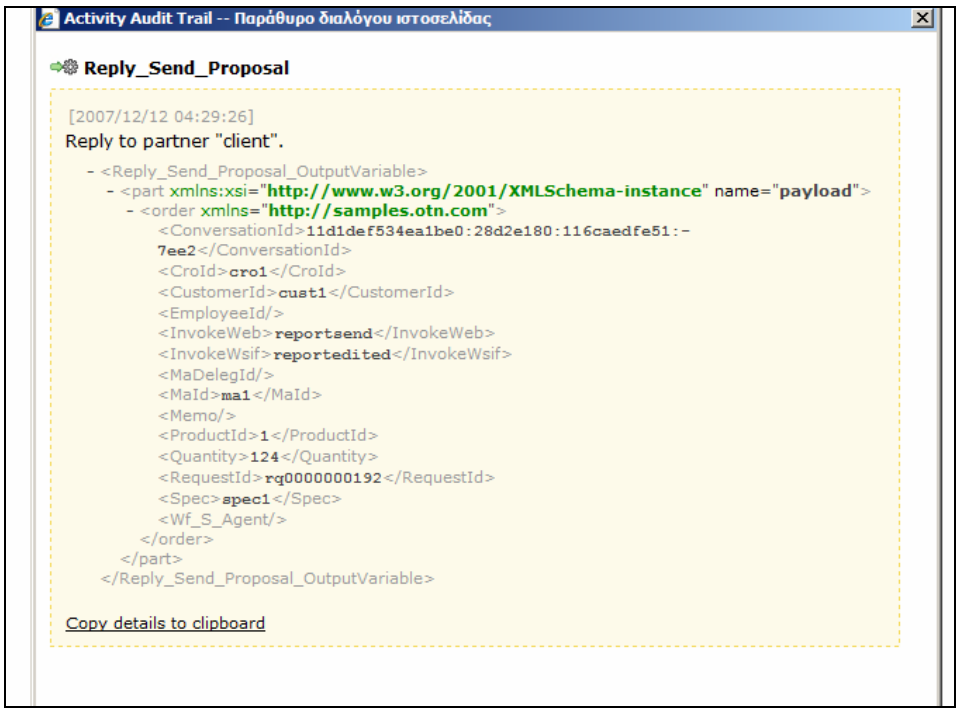
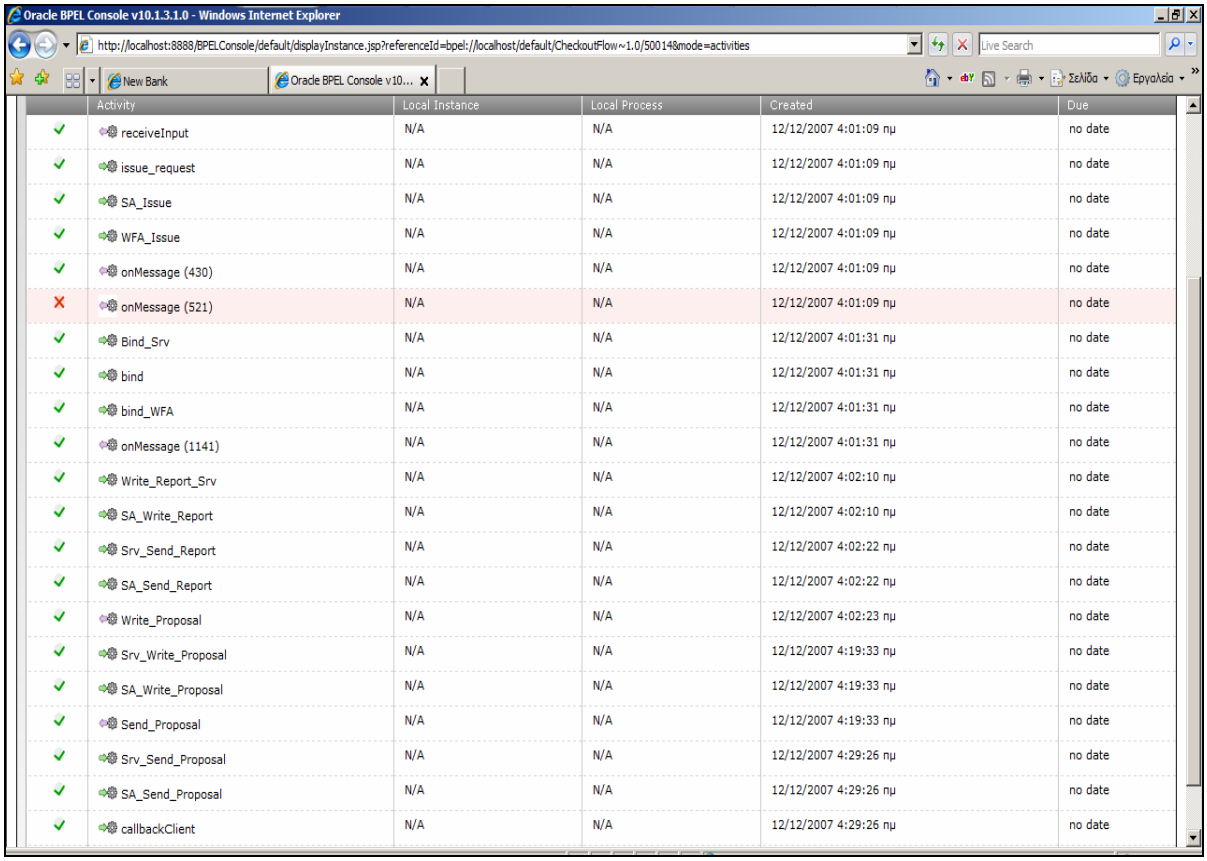


Figure 76. Callback Message

In Figure 77, the operations of the client, workflow and services are presented.



	Activity	Local Instance	Local Process	Created	Due
✓	receiveInput	N/A	N/A	12/12/2007 4:01:09 πμ	no date
✓	issue_request	N/A	N/A	12/12/2007 4:01:09 πμ	no date
✓	SA_Issue	N/A	N/A	12/12/2007 4:01:09 πμ	no date
✓	WFA_Issue	N/A	N/A	12/12/2007 4:01:09 πμ	no date
✓	onMessage (430)	N/A	N/A	12/12/2007 4:01:09 πμ	no date
✗	onMessage (521)	N/A	N/A	12/12/2007 4:01:09 πμ	no date
✓	Bind_Srv	N/A	N/A	12/12/2007 4:01:31 πμ	no date
✓	bind	N/A	N/A	12/12/2007 4:01:31 πμ	no date
✓	bind_WFA	N/A	N/A	12/12/2007 4:01:31 πμ	no date
✓	onMessage (1141)	N/A	N/A	12/12/2007 4:01:31 πμ	no date
✓	Write_Report_Srv	N/A	N/A	12/12/2007 4:02:10 πμ	no date
✓	SA_Write_Report	N/A	N/A	12/12/2007 4:02:10 πμ	no date
✓	Srv_Send_Report	N/A	N/A	12/12/2007 4:02:22 πμ	no date
✓	SA_Send_Report	N/A	N/A	12/12/2007 4:02:22 πμ	no date
✓	Write_Proposal	N/A	N/A	12/12/2007 4:02:23 πμ	no date
✓	Srv_Write_Proposal	N/A	N/A	12/12/2007 4:19:33 πμ	no date
✓	SA_Write_Proposal	N/A	N/A	12/12/2007 4:19:33 πμ	no date
✓	Send_Proposal	N/A	N/A	12/12/2007 4:19:33 πμ	no date
✓	Srv_Send_Proposal	N/A	N/A	12/12/2007 4:29:26 πμ	no date
✓	SA_Send_Proposal	N/A	N/A	12/12/2007 4:29:26 πμ	no date
✓	callbackClient	N/A	N/A	12/12/2007 4:29:26 πμ	no date

Figure 77. Send_Proposal Scope Operations

In Figure 78, the interactions of the client, workflow and services are presented.

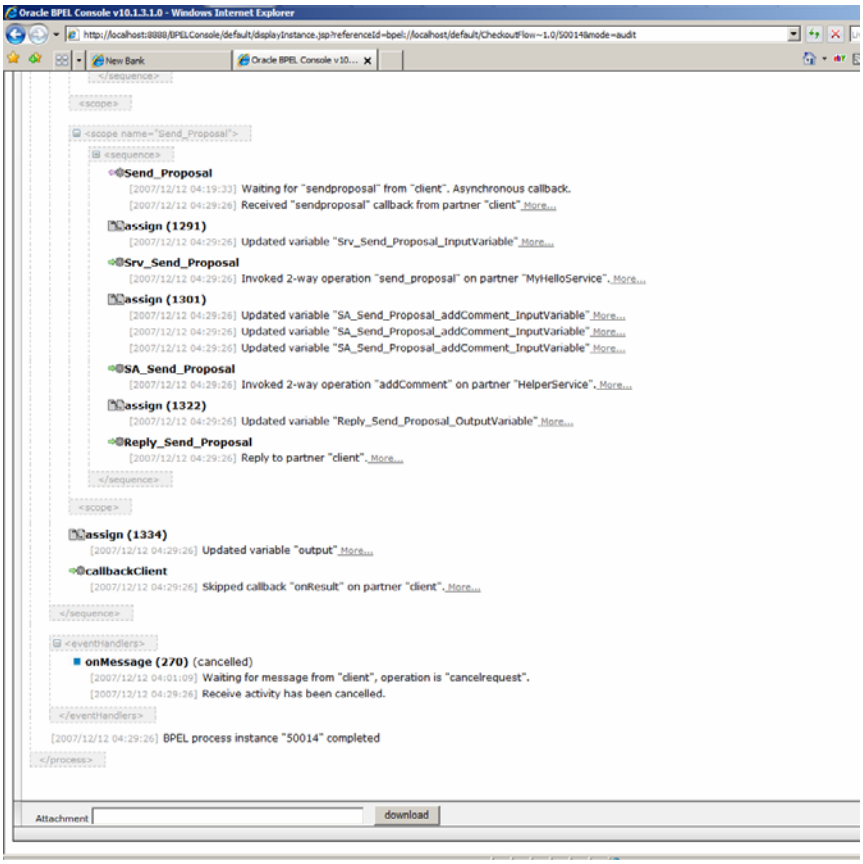


Figure 78. Send_Proposal Scope Interactions

After the Send_Proposal scope, the workflow returns a message to the CRO who issued the request and terminates.

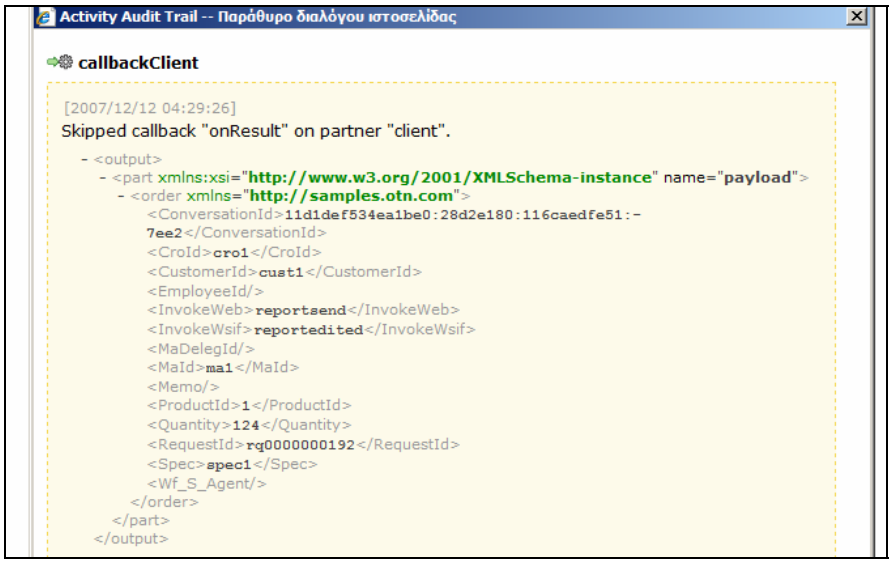


Figure 79. Workflow Callback Message

In Figure 80, the workflow for the Send_Proposal scope is presented.

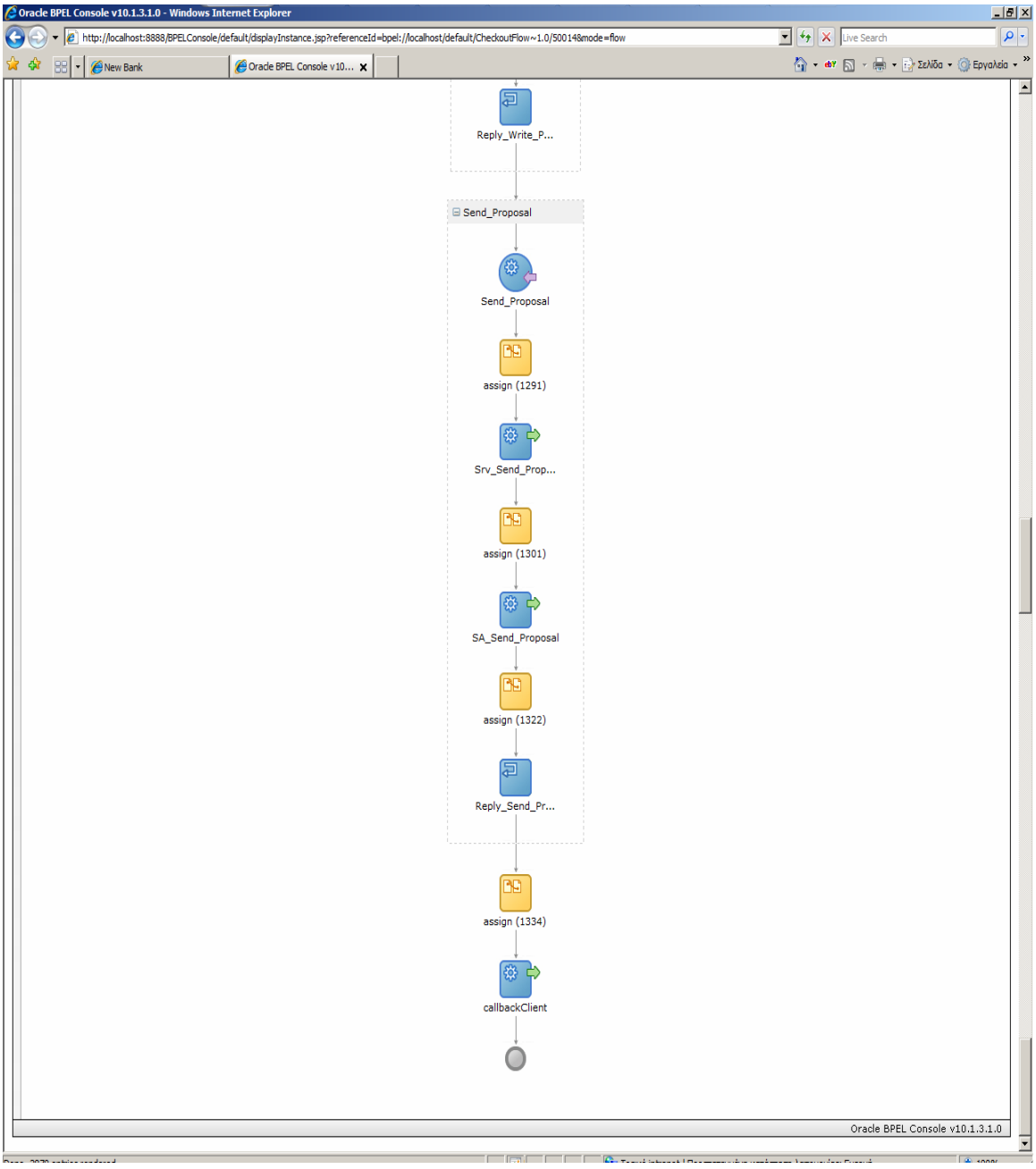


Figure 80. Send_Proposal Scope

References

A Context-Aware Security Architecture for Emerging Applications_ Michael J. Covingtony, Prahlad Fogla, Zhiyuan Zhan, Mustaque Ahamad, College of Computing, Georgia Institute of Technology, Atlanta, Georgia

A role and context based security model, Yolanta Beresnevichiene, University of Cambridge, Computer Laboratory

Access Control to Information in Pervasive Computing Environments, Urs Hengartner and Peter Steenkiste, Carnegie Mellon University

Context Sensitive Access Control, R.J. Hulsebosch†, A.H. Salden, M.S. Bargh, P.W.G. Ebben, J. Reitsma, Telematica Instituut

Context Sensitivity in Role-based Access Control, Arun Kumar; Neeran Karnik, Girish Chafle, IBM India Research Laboratory, Indian Institute of Technology

DW-RBAC: A formal security model of delegation andrevocation in workflow systems, Jacques Wainera,_, Akhil Kumarb, Paulo Barthelmess, Institute of Computing, State University of Campinas

GENERALIZED TEMPORAL ROLE BASED ACCESS CONTROL MODEL (GTRBAC) PART I, Specification and Modeling by James B. D. Joshi, Elisa Bertino, Usman Latif, Arif Ghafoor, Center for Education and Research in Information Assurance and Security, Purdue University, West Lafayette

Human-Centric Network Security Management: A Comprehensive Helper Ghita Kouadri Mostéfaoui, Fribourg University, Switzerland, Patrick Brézillon, Université Pierre et Marie Curie, France

OptIn DBPM: An Information-centric Approach to Business Process Management Rob Mentink, Dirk Wijnker, Manfred Reichert, Hajo A. Reijers

Practical Applications of Triggers and Constraints: Successes and Lingering Issues Stefano Ceri_ Roberta J. Cochrane Jennifer Widomy

Politecnico di Milano IBMAlmaden Research Center Stanford University

Proposed NIST Standard for Role-Based Access Control DAVID F. FERRAIOLO National Institute of Standards and Technology, RAVI SANDHU, SingleSign On. Net and George Mason University, SERBAN GAVRILA, VDG Incorporated and D. RICHARD KUHN and RAMASWAMY CHANDRAMOULI, National Institute of Standards and Technology

Role-Based Access Control Models_yz, Ravi S. Sandhu{, Edward J. Coynek, Hal L. Feinsteink and Charles E. Youmank

Task-Role Based Access Control (T-RBAC): An Improved Access Control Model for Enterprise Environment, Sejong Oh 1, Seog Park, Sogang University, Dept. of Computer Science

Team-and-Role-Based Organizational Context and Access Control for Cooperative Hypermedia Environments Weigang Wang GMD - German National Research Center for Information Technology

Team-based Access Control (TMAC): A Primitive for Applying Role-based Access Controls in Collaborative Environments, Roshan K. Thomas, Odyssey Research Associates Cornell Business and Technology Park

Towards Context-aware Security: An Authorization Architecture for Intranet Environments, Chris Wullems, Mark Looi and Andrew Clark, Information Security Research Centre, Queensland University of Technology

TRBAC: A Temporal Role-Based Access Control Model, ELISA BERTINO and PIERO ANDREA BONATTI, University of Milano, Italy and ELENA FERRARI, University of Insubria, Como, Italy

A Reference Monitor for Workflow Systems with Constrained Task Execution, Jason Crampton, Information Security Group, Royal Holloway, University of London

A Secure Workflow Model, Patrick C. K. Hung, CSIROMathematicalAndInformationSciences, Australia, Kamalakar Karlapalem, InternationalInstituteOfInformationT echnology, India.

Access Control Mechanisms for Inter-organizational Workflow, Myong H. Kang, Joon S. Park and Judith N. Froscher, Naval Research Laboratory, Information Technology Division

Authorization and Access Control of Application Data in Workflow Systems, Shengli Wu, The University of Strathclyde, Glasgow, Scotland, AMIT SHETH, JOHN MILLER, LSDIS Lab, University of Georgia, Athens, Zongwei Luo, IBM T.J. Watson Research Center, Yorktown Height,

Defining The Semantics Of Reactive Components In Event-Driven Workflow Execution With Event Histories, Andreas Geppert, Dimitrios Tombros and Klaus R. Dittrich, Institut fiir Informatik, Universitlt Ziirich, Winterthurerstr

ECA rule-based support for workflows, A. Goh, Y. -K. Koh, Division of Software Systems, School of Computer Engineering, Nanyang Technological University, and D. S. Domazet, Gintic Institute of Manufacturing Technology, Singapore

Enforcing workflow authorization constraints using triggers, Fabio Casati, Silvana Castano and Maria Grazia Fugini, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Dipartimento di Scienze dell'Informazione, Università di Milano

Event-Based Interaction Management for Composite E-Services in eFlow, Fabio Casati and Ming-Chien Shan, Hewlett-Packard Laboratories

Exterminating the Dynamic Change Bug: A Concrete Approach to Support Workflow Change, W.M.P. van der Aalst, Eindhoven University of Technology, Faculty of Technology, and Management, Department of Information and Technology

Managing Workflow Authorization Constraints through Active Database Technology, Fabio Casati, Hewlett-Packard Laboratories, Silvana Castano, Dipartimento di Scienze dell'Informazione-Universit`a di Milano, MariaGrazia Fugini, Dipartimento di Elettronica e Informazione, Politecnico di Milano

Modeling and Analyzing ofWorkflow Authorization Management, Zhang Yi, Zhang Yong and Wang Weinong, Department of Computer Science and Engineering, Shanghai Jiaotong University

Security for Workflow Systems, Vijay Atluri, Rutgers University

SOWAC: A Service-Oriented Workflow Access Control Model, XU Wei, WEI Jun, LIU Yu, LI Jing, Technology Center of Software Engineering, Institute of Software, the Chinese Academy of Sciences

Supporting Conditional Delegation in Secure Workflow Management Systems, Vijayalakshmi Atluri, Janice Warner, MSIS Department and CIMIC Rutgers University, Newark

The Consistency of Task-Based Authorization Constraints in Workflow Systems, Kaijun Tan, Department of Computer Science, University of Pennsylvania, Jason Crampton, Information Security Group, Royal Holloway, University of London, Carl A. Gunter, Department of Computer Science, University of Pennsylvania

Business Process Execution Language for Web Services Second Edition, Matjaz B. Juric With Benny Mathew and Poornachandra Sarang

BPEL Cookbook Best Practices for SOA-based integration and composite applications development, Stany Blanvalet, Jeremy Bolie, Michael Cardella, Sean Carey, Praveen Chandran, Yves Coene, Kevin Geminiuc, Matjaž B. Jurič, The Hoa Nguyen, Arun Poduval, Lawrence Pravin, Jerry Thomas, Doug Todd

SOA for the Business Developer: Concepts, BPEL, and SCA, First Edition byBen MargolisandJoseph Sharpe

Business Process Execution Language for Web Services, Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, Sanjiva Weerawarana

Developing Multi-Agent Systems with JADE, Fabio Bellifemine, Telecom Italia, Giovanni Caire, Telecom Italia, Dominic Greenwood, Whitestein Technologies AG, Switzerland

Workflow Handbook 2005, Layna Fischer

Workflow Management Models, methods and systems, Wil van der Aalst, Kees van Hee

Workflow Modeling—Tools for Process Improvement and Application Development, Alec Sharp and Patrick McDermott

Designing Web Services with the J2EE™ 1.4 Platform JAX-RPC, SOAP, and XML Technologies, Inderjeet Singh, Sean Brydon, Greg Murray, Vijay Ramachandran, Thierry Violleau, Beth Stearns

SOA Using Java™Web Services, Mark D. Hansen

Enterprise Service Oriented Architectures Concepts, Challenges, Recommendations, JAMES MCGOVERN, OLIVER SIMS, ASHISH JAIN, MARK LITTLE