Agent-based Mobility add-in feature for Object Transaction Service (OTS)

Abstract: *Service session mobility is a new concept in the next telecom service generation. It allows users to move from one terminal in one network to another terminal in another network with service continuity. This concept requires to migrate the current service execution related information, including service context, service data, etc, from one terminal to another. For services that execute within transactions, transaction context is a part of service context and must be migrated too. This paper presents our work on this problem, based on CORBA Object Transaction Service. Mobile agent is the enabling technology which enhances the transaction service with mobility features.*

This model was initially designed for VHE architecture of VESPER, a European project to design a model for Virtual Home Environment in the next generation of telecom network. However, it can be applied to any other architecture, which require to solve the problem of service session mobility with transaction.

1. Introduction

Should Internet electronic commerce meet user mobility requirements, one of the greatest challenges would be to maintain the user service execution while allowing enough flexibility to provide the user with a sensation of comfort and liberty. Current mobility management in telecom networks such as GSM mainly concentrates on *terminal mobility* which means that the user can continuously use a service while moving with his/her terminal. Next generation telecom networks such as UMTS ([1]) or TINA ([2]) extend the mobility features with *session mobility* concept which allows the user to change his/her terminal during the service execution. Typically, the user suspends the service from one terminal and resumes this service from another. For services which require a client application, service data, and, more generally, a service context in the terminal, the Distributed Platform Environment (DPE) which supports the service must provide enough capabilities for this service execution related information to migrate from the old terminal to the new one.

Since distributed transactions play a very important role in the electronic commerce domain, this paper presents our work on the possibility to use distributed transactions to support the next generation telecom services. Currently, commercial domains already use distributed transaction platforms which implement different transaction specifications. Object Transaction Service - OTS ([3]) - is the specification for distributed transaction in the CORBA environment. Since CORBA is selected as a de-facto DPE for some telecom service architectures, we consider OTS as the implicit common transaction environment for the next generation telecom service platforms. Because the mobility aspect was not taken into account when the OTS specification was designed, one of our goals is to enhance the transaction service with a mobility feature which can be used in the next generation of telecom architectures.

Several ongoing research approaches define some new distributed transaction models such are *Multi-Agent Cooperative Transaction* ([4]) or *Kangaroo* ([5]). We propose to maintain and use the standard OTS model and associate to it the tools which could make it meet the user mobility requirements. Given its autonomy, adaptability and reactivity properties, mobile agent technology is used as an enabling mechanism. In our model, agents do not directly participate to the transaction but they interact with the existing OTS platforms in order to migrate, when necessary, parts of the transaction environment.

Our proposition is that the new mobility add-in feature respects the three following objectives:

- It should be transparent to the application execution;
- It must not reduce nor interfere any of the OTS features provided by the existing specification;
- Extension, if any, to the existing OTS specification must be minimized.

Our research is currently applied to VESPER ([6]), a European project that defines a Virtual Home Environment (VHE) architecture for the next generation telecom services. We believe that VESPER architecture is a very demanding platform for our research and therefore, section 2 introduces some VESPER design features related to our work. A typical value-added service, the meeting arrangement service, namely *Calendar*, illustrates how a distributed telecom service can benefit from transactions for maintaining the agendas consistent.

Section 3 introduces an overview of OTS model, then focus on the OTS deployment aspect.

The add-in mobility feature for OTS model is introduced in section 4. The motivation for using mobile agents is argued, then the detail architecture of OTS add-in mobility feature, based on mobile agent technology, is presented.

Section 5 describes an implementation prototype using MaaoOTS ([7]), an OTS platform developed in our laboratory and Grasshopper ([8]), a mobile agent platform of IKV. Several demonstration scenarios are introduced and the time needed for agent migration is evaluated.

Finally, we conclude with the perspectives open up by our work.

2. VESPER Model and Service Provision

As introduced above, the objective of VESPER is to define a VHE architecture for the next generation telecom services. The 3rd Generation Partnership Project (3GPP) consortium defines VHE as *"a concept for personal service environment portability across network boundaries and between terminals. The concept of the VHE is such that users are consistently presented with the same personalised features, User Interface customisation and services in whatever network and whatever terminal (within the capabilities of the terminal and network), where ever the user may be located."* ([9])

Stemming from the VHE idea, the concept of "Service Session Mobility" has been analyzed in VESPER and is detailed hereafter.

2.1. VESPER distributed roles and Service Session Mobility

Figure 1 illustrates three distributed roles and their interfaces as extracted from the VESPER business model ([10]). Several other VESPER roles, which do not directly interfere with our work on OTS add-in mobility feature, are not showed here. The *Consumer* role represents the end-user who access and uses the service. *VASP* (Value Added Service Provider) represents the service provider role, which implements the service and offers its usage to the user.

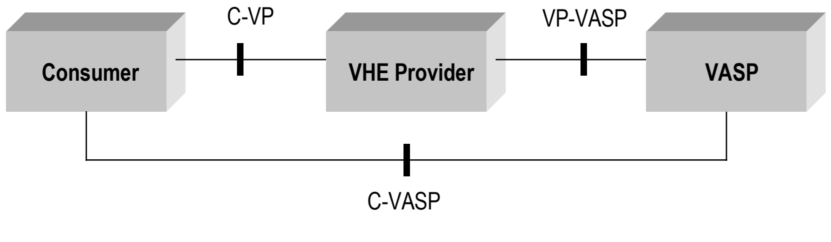

Figure 1: VESPER distributed roles

The third and most important role in the VESPER business model is the *VHE Provider*. It takes the responsibility to:

- provide discovery and access to the service,
- adapt the service interactions with the user depending on the user terminal capabilities and on the user interface preferences,
- control the service session mobility.

The *VHE Provider* is structured in several VHE components such as *Session Manager*, *Adaptation Manager*… It also includes some utilities components, among which a *Transaction Manager*. All components externalize several interfaces to the *Consumer* and to the *VASP*, according to usage rules. Within the limitations of this document, we focus on the *VHE Session* component because it shows the principle of the session mobility control.

The *VHE Session* component provides to the user all necessary operations to start, quit, suspend, and resume a service. Figure 2 illustrates this control and expands the optional functions, which can be performed by the *VASP* and *Session* component as far as data management is concerned.

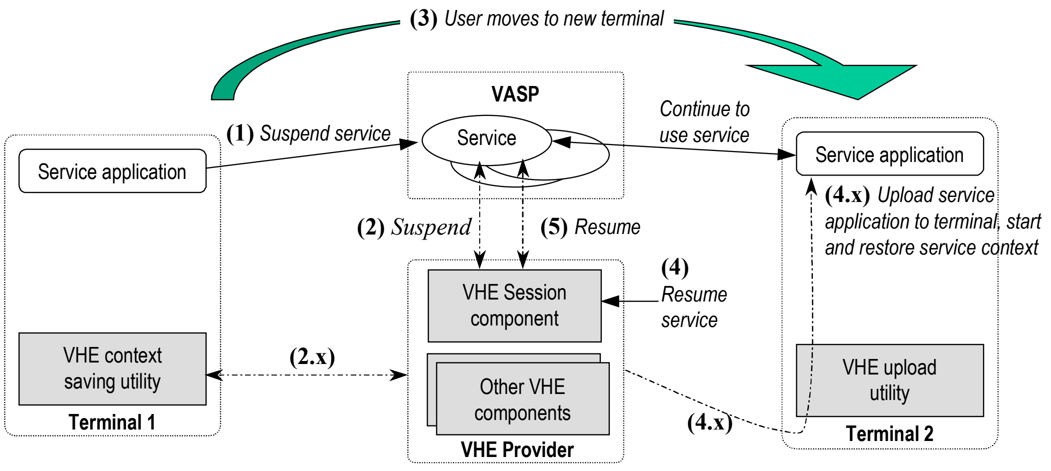

Figure 2: Service Session Mobility in the VESPER model

Step 1: The user requests to suspend his service. Because the terminal may have service related data, which are not saved in a stable memory (the terminal may even have no stable memory), the *VASP* may save this data in its local storage.

Step 2: The *VASP* notifies the *VHE Provider* of the user suspension The *Session* component may take any disposition to save the service context, including, for example, the user suspended state, references of the client application objects in the terminal, possibly terminal states with regards to the service (specifically in the case when transactions are used as will be seen later). These procedures necessitate to have a *VHE saving utility* on the suspending terminal (step 2.x).

Step 3: After suspension, the user moves to a new terminal

Step 4: The user requests to resume his service by using a standard access support from *VHE components*. The resumed terminal may not have the necessary service application although it can support. In this case, a new service application needs to be uploaded in this terminal and started with the service execution related information saved at suspension time. These procedures necessitate to have a *VHE uploading utilities* on the resuming terminal (step 4.x).

Step 5: Finally, the *VHE Provider* contacts with *VASP* to resume the service.

2.2. Calendar Service

The Calendar service is a Value Added Service used to validate the VESPER model. This service provides a coordinating environment allowing multiple users to set up their meetings. Figure 3 presents the Calendar service basic usage scenario.

Step 1: When a user wants to propose a meeting with some others colleagues, he submits a meeting request to the Calendar service. This request contains all necessary meeting information such as time, date, location... and a list of users (participants) who should participate to the meeting.

Step 2: Calendar service requests *VHE Provider* to invite the participants to the meeting preparation procedure.

Step 3: The *VHE Provider* then performs several "network" tasks to invite the participants, process their joining the service, and finally, notifies the Calendar service that participants join.

Step 4: The Calendar service indicates the meeting request to the joining participant.

Step 5: Each participant replies to the meeting request by proposing his convenient time slot.

Step 6: After having received replies from all participants, the Calendar service calculates the final possible meeting time and sends it back to the requester for approval. This user can decide to accept the meeting solution or to reject it. In the case where the meeting solution is accepted, all private agendas, which are kept in the user and participants' terminals, are updated.

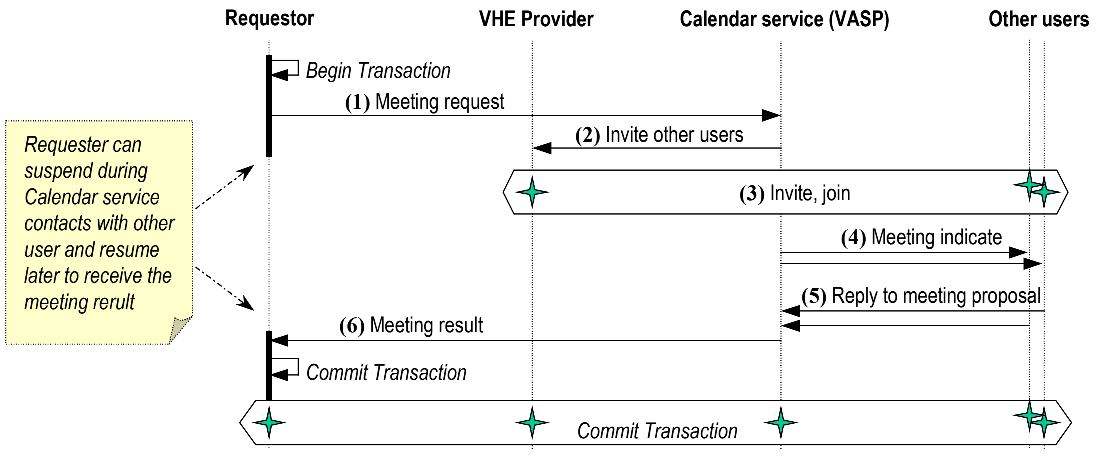

Figure 3: Calendar service usage scenario

Before sending the meeting request, in order to guarantee the consistency and the isolation of the users' agendas, the requester application starts a transaction and selects the requester's agenda time slot for the meeting. The effect is to lock the time slot until the transaction is completed. Participants' proposed time slot are also selected within the transaction scope. The effect is that their proposed slot times are also locked until the meeting is resolved. When the requester receives and accepts the meeting result (step 6), he validates the transaction so that the meeting time is securely saved on all agendas and all locks are released. For the sake of performance, this basic scenario can be enhanced by using nested transactions ([11]), but this would go beyond the paper scope.

Invitations and join process can take quite a long time. The meeting requestor may not want to remain online until all participants have joined and responded because he has to pay for the network connection. Session mobility allows the requestor to suspend his participation and to resume later in order to receive the meeting result. Likewise, a participant who has joined promptly may suspend and resume later to get the meeting result. This scenario requires the transaction environment, which is established in one terminal, to be migrated to a new terminal so that, for example the meeting requestor can validate the transaction after that he accepts the meeting result in a new terminal

2.3. Conclusion

We have shown that the VHE provision for session mobility implies that the *VHE Provider* must be able to upload information from the terminal at suspension time, and to download applications and information into the terminal at resuming time.

The Calendar service presents an example where service user data are distributed and should be maintained consistent despite failures and concurrent requests. The distributed transaction service is proposed as a tool to guarantee the agendas consistency. The next section presents in detail the transaction service deployment in order to identify the type of information that the *VHE Provider* shall be able to upload/download from/in the terminal when offering a transaction support.

3. Use of the Object Transaction Service

3.1. OTS Overview

Object Transaction Service (OTS) supports the concept of a distributed transaction, that is, it guarantees benefits of the following properties: Atomicity, Consistency, Isolation and Durability. Atomicity means that all or none of the data manipulated in the unit of work are validated. Consistency implies that the unit of work moves the data from a consistent state into a new consistent state. Isolation means that the unit of work intermediate states are not visible to other unit of works. Durability means that the effects of a validated transaction are persistent.

From the usage point of view, OTS offers four roles, namely those of a *Transactional Client*, a *Transactional Server*, a *Recoverable Server* and a *Transaction Manager*.

- A *Transactional Client* is an arbitrary program that begins a transaction, invokes operations on transactional servers and finally, validates or aborts the transaction.
- A *Transactional Server* is a program whose behavior is affected by being invoked within transaction scope. It contains or indirectly refers to persistent data that can be modified by the transactional client requests.
- A *Recoverable Server* contains one or several *Recoverable* objects, whose data are modified within the transaction. As a result, a recoverable object must participate in the transaction validation or abort. It does so by registering a *Resource* object with the *Transaction Manager*. Typically a recoverable object can be composed of a database and a resource object.
- The *Transaction Manager* drives the validation and abort protocols by issuing appropriate requests to the *Resource* objects registered in the transaction.

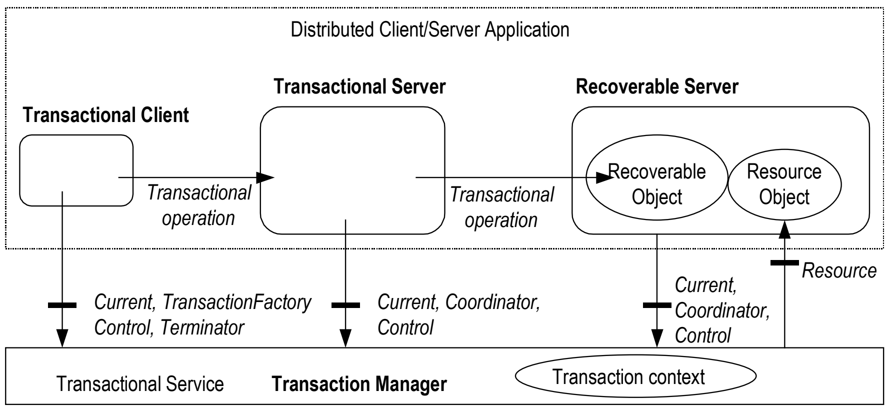

Figure 4: CORBA Transaction Service

Transactional Client, *Transactional Server*, *Recoverable Server* and *Transaction Manager* can be located in different process or machines. They communicate through the CORBA Object Request Broker (ORB), over an IP network. In the context of telecom services, typically, the *Transactional Client* can be located in a terminal, for example the Calendar requestor's and participants' terminals. The *Transactional Server* and *Recoverable Server* can be located in a *VASP* and the *Transaction Manager* in the *VHE Provider*.

Transaction Context is an abstract concept, which defines the transaction scope. It is shared by the objects that participate to the current transaction. A transaction context is created when a transactional client starts a transaction and it is destroyed when the transaction completes. The transaction context is propagated in each operation invoked on a transactional server or on a recoverable server so that they can be associated to the client transaction context

3.2. OTS Application Programming Model and OTS Library

The most normal way for an application to use a CORBA service, and OTS in particular is to invoke the *Current* object. In the *indirect* mode, OTS offers on the *Current* object the start, validate, and abort transaction operations as well as other operations not used in this paper. The hierarchical objects implementation in the *Transaction Manager* is completely transparent to the application. OTS supports another mode, namely the *direct* mode. In this mode, the application has to be more or less aware of the hierarchical objects implementation in the transaction manager so that it can look for and invoke operations on the correct object, for example start transaction on the *Transaction Factory* and validate transaction on the *Terminator* object.

Another aspect in executing a distributed transaction is the transaction context propagation. The common way for an application is to use the *implicit* mode. In this mode, the transaction context is implicitly added to each invocation between the client and server application. At the server side, upon receipt of the invocation,

- The transaction context is suppressed from the CORBA object request,
- The transaction context is automatically associated with the current application thread,
- A particular *Resource* object can be created and automatically registered to the *Transaction Manager*.
- Finally, the CORBA object request is delivered to the server.

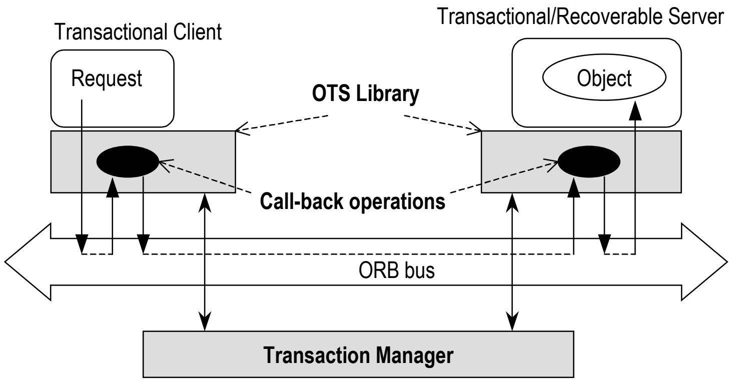

Figure 5: Transaction Library and Callback functions

Both the *indirect* API mode and the *implicit propagation* feature are provided by the OTS library in all OTS implementation products. The OTS library is configured to cooperate with CORBA environment and filter each CORBA invocation. *Interceptor* is the mechanism that most OTS products use to implement the callback functions. Each time a CORBA request goes from a client to a server, callback functions are executed and manage the transaction context implicit propagation.

There is another propagation mode, namely the *explicit* mode. In this mode, it is the application responsibility to perform all functions described above. Clearly, in a VHE context where one of the main objectives is service portability, we consider that service applications should not be concerned with OTS internal objects nor with transaction context propagation. Therefore only the *indirect* and *implicit* modes are taken into account further in this paper.

3.3. Conclusion

Use of the transaction service implies for the client and server applications to access to the OTS library. As was introduced in section 2, our scope is to enable the use of OTS by a user who may suspend his service on one terminal and resume the service on another terminal. The user can resume on a terminal which may not contain the OTS library. There is a need to download this library in the resumed terminal.

Further, we have seen that, once a transaction is started, the OTS library maintains a transaction context for this transaction. Additionally the library callback functions may create other specific information in order to control the invocation replies before to authorize the client to validate the transaction. The transaction context and this information constitute a library state. When the user moves from one terminal to another, this state must be transferred on his resumed terminal.

4. Use of Agent technology

4.1. Motivation

We propose a general model to migrate the OTS library and its state from one terminal to another. In principle, when the user suspends a service, the library state is retrieved from the terminal and saved in the *VHE Provider*. When the user resumes on another terminal, the library is downloaded and the state saved in *VHE Provider* is restored in the resumed terminal. Theoretically, this model can be concretized by several mechanisms like traditional client-server computing. However, there are two main reasons, according to the seven good agent capabilities presented in [12], which make mobile agents technology as the most challenging solution:

- Restoring the OTS library state in a terminal requires a specific software to be preinstalled. This implies that procedures consisting of uploading and installing this specific software must be done before restoring the OTS library state. These procedures should be generic enough in order to be executable in as many as possible terminal types. Mobile agent, with the capabilities of "dynamical adapting" ([12]), and with very limited requirements on the hosting system (thank to the agent platform), is a very attractive solution.
- Secondly, according to the OTS library state migration model described above, the procedures of retrieving transaction context from a user terminal and restoring it in a new one require several interactions between the *VHE Provider* and user terminals. Due to the network latency, these interactions may introduce delays at both suspension and resumption time. With mobile agent technology, an agent can migrate to a user terminal as a representing of *VHE Provider* and perform autonomously all necessary tasks through local interactions, then come back to *VHE Provider*. This is the agent's advantage to "overcome network latency" ([12]). Of course, we have to take into account the agent migration time itself.

From the above analyses, mobile agent technology has been chosen to realize our add-in mobility feature in OTS. It is presented in detail in the following sections. The last section in this paper will show our calculation of the agent migration time depending on different network technologies.

4.2. OTS Library Mobility

4.2.1. Agents roles

The mobility add-in architecture for OTS is presented in figure 6, based on two agents, the *OTSAgent* and the *OTSSessionManagerAgent.*

OTSAgent is a mobile agent, which migrates to the user terminal whenever the user starts a service declared as using transactions, for example the Calendar service. The main responsibility of *OTSAgent* is to bring the OTS library to the user terminal to install it and to stay in the user terminal, in order to locally manipulate the transaction context when necessary.

At service suspension time, *OTSAgent* retrieves the transaction context from the OTS library and comes back to *VHE Provider*.

At service resumption time, this *OTSAgent* moves again to the user terminal, not only with the OTS transaction library but also with the transaction context. After deploying the OTS library in the user terminal execution environment, *OTSAgent* imposes also the transaction context to the client application thread.

Within a given service, as in Calendar service, there is one *OTSAgent* for the requestor of the meeting, and one for each the meeting participants, enabling autonomous all users to suspend/resume concurrently.

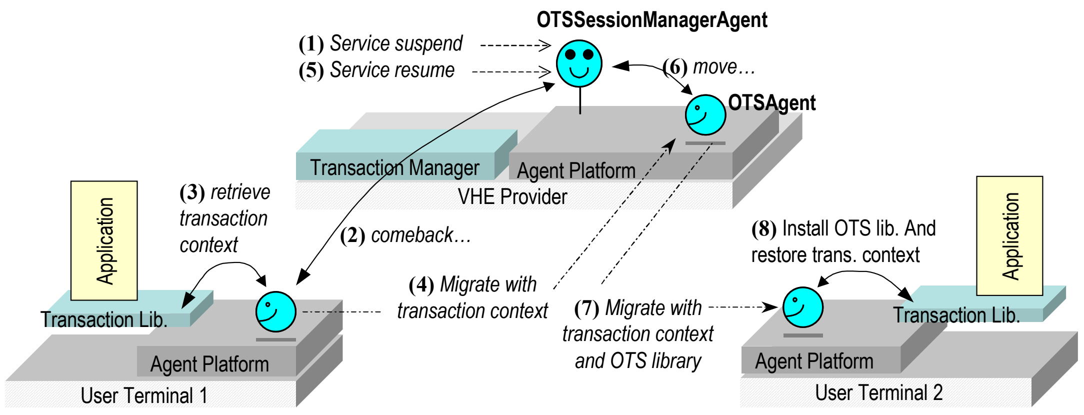

Figure 6: Mobility add-in architecture for OTS

The *OTSSessionManagerAgent* is a stationary agent, which resides in the *VHE Provider*. It is shared by all services that use transactions. This agent shall be notified of users suspension and resumption by the *VHE Session* component and control the *OTSAgent* migration.

Figure 6 presents the sequence of actions executed when the user suspends and resumes on a new terminal.

Step 1: *OTSSessionManagerAgent* receives a notification about service suspension.

Step 2: It orders the *OTSAgent* in the suspending user terminal to come back to the *VHE Provider*.

Step 3: This *OTSAgent* retrieves the transaction context from the local OTS library

Step 4: *OTSAgent* migrates to the *VHE Provider* with the transaction context.

Step 5: The user requests to resume the service from a new terminal. *OTSSessionManagerAgent* receives this user resumption event

Step 6: It orders the *OTSAgent* to move to the terminal.

Step 7: *OTSAgent* moves to the user new terminal with the OTS library and with the transaction context. In the case where the OTS library is already installed in the current user terminal, *OTSAgent* brings only the Transaction context.

Step 8: *OTSAgent* installs the transaction library in the user terminal and restores the transaction context. It then stays in the new user terminal to wait for another command from *OTSSessionManagerAgent*.

4.2.2. Extensions on the Current Interface

In order to enable *OTSAgent* to retrieve and install the transaction context in the OTS library, we propose to extend the *Current* interface with two new operations:

```
interface Current {
  void export_transaction_context(out sequence<octet> tr_ctx)
     raises(NoTransaction);
  void import_transaction_context(in sequence<octet> tr_ctx)
     raises(InvalidTransactionContext);

  ... // other standard operations
}
```

When invoked, *export_transaction_context()* converts the current application transaction context into a string of bytes and provides it as the "out" parameter. If there is no transaction currently associated with the application, the *NoTransaction* exception will be raised. *import_transaction_context()* has one "in" parameter which is a string of bytes. When invoked, *import_transaction_context()* restores the string of bytes into a transaction context, and installs it in the OTS library initial structure. If there is an exception this configuration, an exception *InvalidTransactionContext* will be raised.

Thus, there is a mandatory requirement for OTS implementation. The transaction context must be able to be converted into a string of bytes and again, from a string of bytes. It is called S*treamable* in CORBA, or *Serializable* in Java. Although it is not strictly required but a good way is that OTS products can use CORBA Externalization service to specify their *Streamable* transaction context. An empty interface named *TransactionContext* is defined which inherits from CORBA *Streamable* interface:

```
interface TransactionContext : Streamable {
}
```

To convert the transaction context into or from a string of bytes, OTS products can implement two operations *externalize_to_stream()* and *internalize_from_stream()* as defined in the *Streamable* interface.

Note that there is no constrains in the transaction context implementation and the way to associate it with the application thread. They are still completely up to the implementation choice.

4.3. Resource Objects Mobility

4.3.1. User data location

In the Calendar service, as presented in section 2, the meeting requestor as well as the participants agendas are located in their terminals. Agendas are maintained consistent by using transactions. This implies that part of the application in the terminal plays the role of an OTS *Recoverable* server and control access to the agendas. We focus on the case where the *Recoverable* server and its data move from the suspended terminal to the resumed terminal. As discussed below, this is likely to be the case in the near future, using more and more sophisticated SIM cards (currently GSM network user's address book can already be kept in their SIM card.).

Our interest hereafter focuses on the registration aspects: As seen in section 3, *Recoverable* servers register a *Resource* object to the *Transaction Manager*. Upon transaction completion the *Transaction Manager* propagates the commit/abort information to all registered *Resource* objects. Therefore, if the user suspends the service and resumes from a new terminal, the new address (Interoperable Object Reference - IOR) of the *Resource* object must be made known *Transaction Manager*.

The implementation of a *Recoverable* server is completely up to application, all OTS knows about is the *Resource* object. We focus on the case where The *Resource* object resides in the OTS library.

This case applies particularly well to relational database such as Oracle, Informix, which offer the so-called standard XA interface in order to participate to distributed transactions. The *Resource* object (figure 7) is the intermediate between the *Transaction Manager* and the XA interface and, in particular, maps the *Transaction Manager* invocations onto appropriate XA functions. Typically, this mapping is provided by the OTS library in the *implicit propagation* mode.

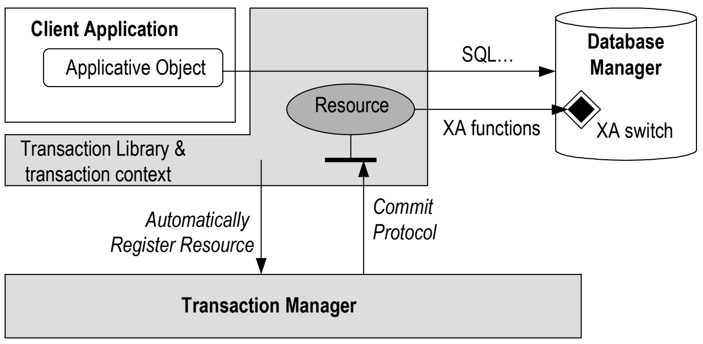

Figure 7: OTS Library internal Resource object and XA Database

The OTS library callback function creates and registers the *Resource* object to the *Transaction Manager* when it receives the first invocation related with a transaction.

This design was already implemented in our MaaoOTS ([7]) product and tested with Oracle database. It is currently under discussion in OMG OTS group to make it a part of OTS specification.

We propose hereafter to extend the *Transaction Manager* in order to support the *Resource* object migration.

4.3.2. Extension on the Transaction Manager Interface

OTS specification provides the *register_resource()* operation on the *Transaction Manager Coordinator* interface to register a *Resource* object. We propose to extend the *Coordinator* interface with a new operation, namely *suspend_resource().* This operation will be invoked at service suspension time to notify the *Transaction Manager* that an existing resource is moving. At resumption time, the *Resource* object will be registered using the existing *register_resource()* operation.

```
interface Coordinator {
  void suspend_resource(in Resource r)
     raises(Unavailable);

  ... // other standard operations
}
```

Suspending a *Resource* is linked to the suspension of the terminal, therefore we propose that, when the *OTSAgent* invokes *export_transaction_context()* on the *Current object,* this object invokes *suspend_resource()* on the *Coordinator* interface to deregister all *Resource* objects. All information related to the *Resource* object state (specific states are used during the validation protocol for recovery purpose) can be returned in the *export_transaction_context()* "out" parameter together with the transaction context. When *OTSAgent* migrates back to the VHE Provider, all these information are taken to the *VHE Provider*.

At the resumption time, when OTS library receives the *import_transaction_context()* invocation from *OTSAgent*, and after having restored the transaction context, it invokes the *register_resource()* operation to register the *Resource* object, with new address, on the *Coordinator* interface.

Note that in the case where the application itself implements the *Resource* or use *Resource* objects provided by another application (third-party application) such as JDBC XAResource, it is the responsibility of the application to register its *Resource* object and design its mapping on the data management system. Our proposal applies as well, however it will also be the responsibility of the application to deregister its *Resource* object

5. Implementation, Demonstration and Evaluation

As discussed above, moving OTS transaction library with the transaction context over network may take long time. In order to evaluate the above mobility feature, we had implemented it with MaaoOTS and Grasshopper. This section presents the implementation with two demonstration testing scenarios. Based on that, we estimate the needed time of *OTSAgent* migration in the several future telecom networks.

5.1. MaaoOTS prototype and Grasshopper

MaaoOTS is an OTS product developed in our laboratory. It contains an OTS *Transaction Manager* and OTS library for C++ and Java language. This library uses CORBA interceptor mechanism to receive the callback functions from CORBA bus in order propagate transaction context between *Transaction Client* and *Transaction Server* ([13]). MaaoOTS library also provides a particular *Resource* object, named *RODB*, which allows application to work with XA Database. Currently, MaaoOTS can work with Orbix and VisiBroker, the two popular CORBA products.

Grasshopper is a mobile agent platform that supports CORBA communication. It is the only one product, which conforms to MASIF standardization now. VisiBroker and Orbix are the two CORBA environment that Grasshopper supports, that's why we selected Grasshopper for developing our model.

5.2. *Demonstration - Basic scenario*

Figure 8 illustrates the deployment used for our test, based on the Calendar service:

- One PC WinNT 4 with Grasshopper agent platform plays the meeting requestor terminal role.
- One Solaris station with Grasshopper agent platform plays the *VHE Provider* role including provision of the OTS service (MaaoOTS).
- A second Solaris machine supports the Calendar server and has an Oracle database.

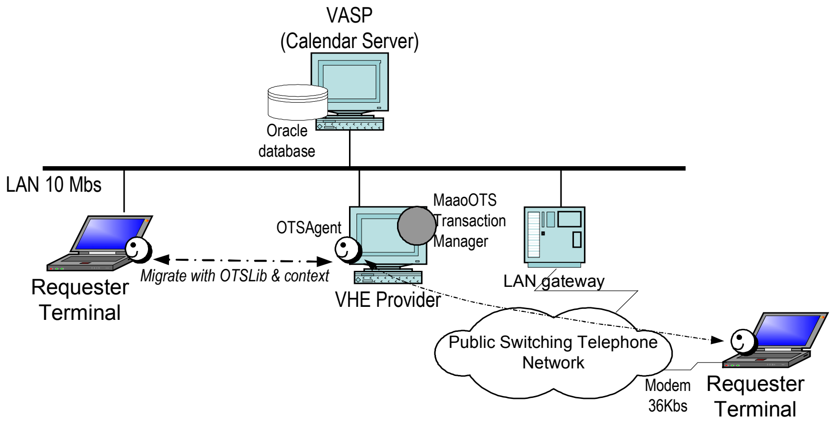

Figure 8: Evaluation scenario in LAN and PSTN

5.3. *Evaluation Results and Calculations*

The MaaoOTS library size is 164Kb. The transaction context size depends on each OTS product but normally it does not exceed 1Kb. So, we can consider that the total amount of data which *OTSAgent* has to migrate over network, is about 165Kb. The *OTSAgent* size itself is 5Kb.

The first test used 10 Mps LAN network. In the second test, the *OTSAgents* migrates in a PSTN, with a phone line of 36Kbs. In both test cases, *OTSAgent* migrates from the suspended terminal to the *VHE Provider*, then to the resumed terminal with MaaoOTS transaction library and transaction context.

By measuring the *OTSAgent* migration time in the two above testing scenarios, we have that for the first test in LAN 10Mbs, *OTSAgent* takes 4 seconds to upload MaaoOTS library and install it to the user terminal. For the second test with PSTN 36Kbs line, it takes 75 seconds. Based on these measured information, we have calculated the time needed when *OTSAgent* migrates in different networks[1]. Figure 9 presents the function graph, which shows the relation between network bandwidth and *OTSAgent* migration time.

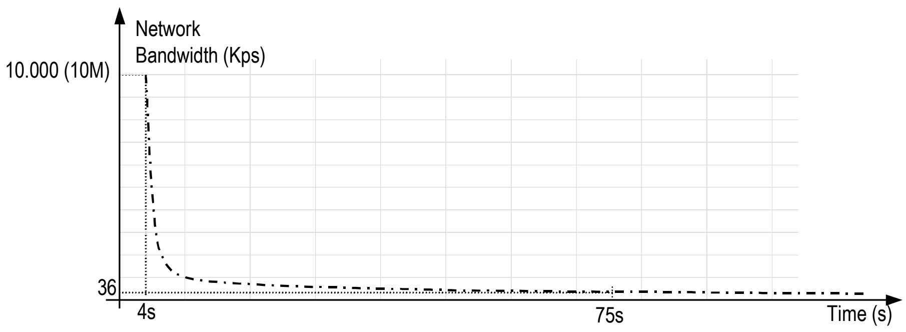

Figure 9: Relation between network bandwidth and *OTSAgent* migration time

[1] The information about future networks speech was taken from "Universal Mobile Telecommunications System (UMTS) Protocols and Protocol Testing" - http://www.iec.org/online/tutorials/umts/

Network type	**Bandwidth** (bps)	***OTSAgent* migration time** (second)
Fast Ethernet LAN	100.00M	3.7
Ethernet LAN	10.00M	4
ISDN	54.00K	51.2
PSTN 36K	36.00K	75
PSTN 56K	56.00K	52
GSM	9.60K	270.9 (4.5 minutes)
GSM 2+	57.60K	48.2
GPRS	171.20K	18.7
EDGE	553.60K	8.4
UTRA(UMTS)	1920.00K	5.1

The above table shows that *OTSAgent* migration time in current PSTN network with modem 56K is 52 seconds, quite long but also acceptable for user. For the current GSM users with the bandwidth of 9.6K, the time needed is about 4 minutes and 30 seconds, an unacceptable time. However, for the future telecom networks, for example UMTS, the time is significantly smaller and becomes acceptable.

5.4. Extended Demonstration Scenario - VHE Model validation

Figure 9 presents the complete scenario of the Calendar service, using the OTS mobility add-in feature, within VHE context. The main objective of this scenario is to show the link between the user, the application and the add-in feature to make a meeting request, then suspends in order to resumes from another terminal and validates the meeting solution.

Step 1-2: The user starts Calendar service. The Calendar application maybe uploaded to the user terminal. The interaction between user and service is controlled by *Adaptation Manager*, a component in *VHE Provider*.

Step 3-6: The *OTSSessionManagerAgent* is notified about this service starting event. It creates an *OTSAgent* and request to move to user's terminal. The *OTSAgent* migrates with MaaoOTS library, then installs to the user terminal. It configures the MaaoOTS library as "null transaction context" meaning that currently there is no transaction context in *OTSAgent*.

Step 7-11: When the user enters a meeting request, the application looks for the *Current* object in MaaoOTS library. Being configured as "null transaction context", MaaoOTS library creates a new object *Current* and returns its reference to the application. The application starts transaction by invoking the *begin()* operation in this *Current* object. As the consequence, a new transaction context is created in MaaoOTS library.

Step 12-16: After a while, the user requests to suspend. *OTSSessionManagerAgent* receives this event from *VHE Session* component, then orders the *OTSAgent* to come back to the *VHE Provider*. *OTSAgent* invokes *export_transaction_context()* on MaaoOTS library to get the transaction context and moves to *VHE Provider*.

Step 17-21: The user requests to resume from a new terminal. *OTSSessionManagerAgent* receives this event from *VHE Session* component, then orders *OTSAgent* to move to the user terminal. This time, *OTSAgent* migrates with MaaoOTS library and the transaction context, then installs to user terminal. It configures the MaaoOTS library as "not null transaction context" meaning there a transaction context currently in *OTSAgent* and registers the Resource objects to the *Transaction Manager*.

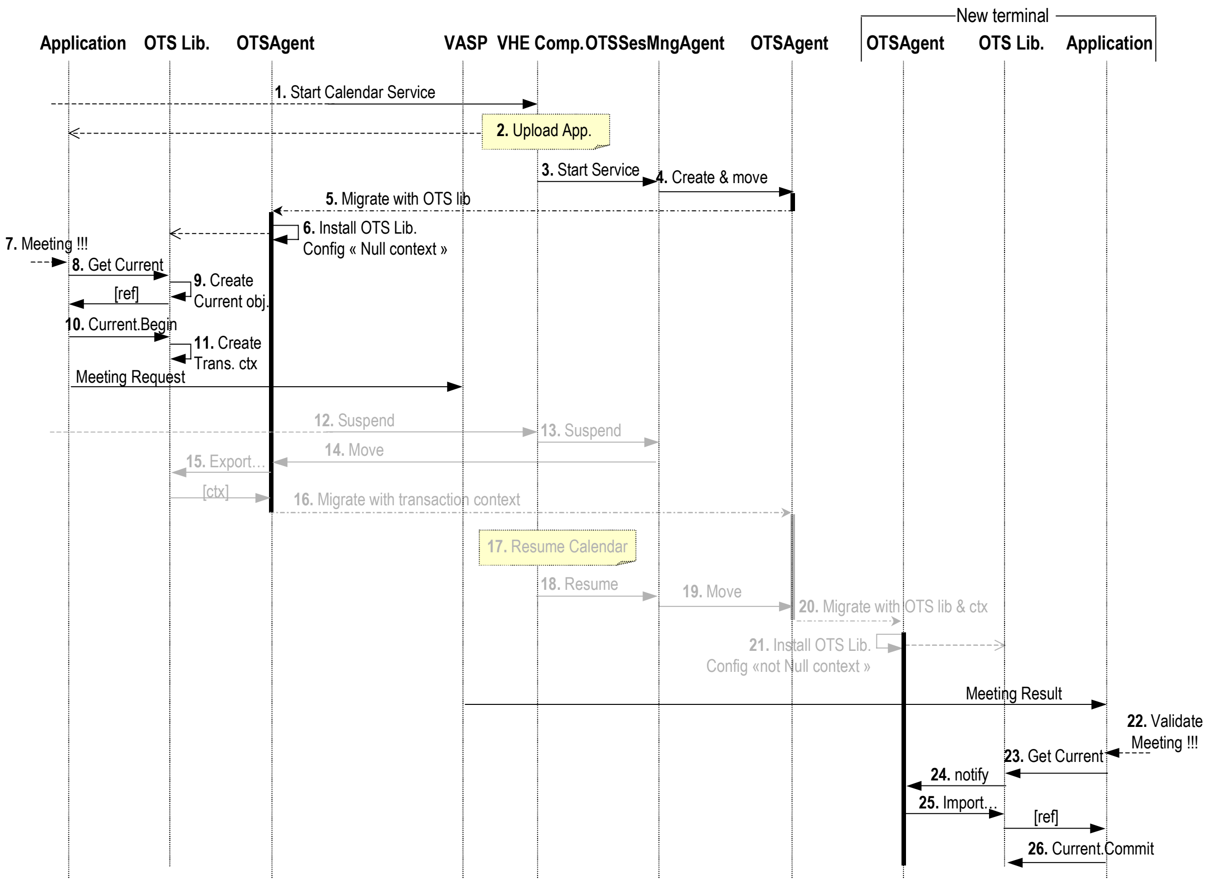

Figure 10: End-to-end scenario of OTS mobility feature in VHE model

Step 22-26: Finally, the user receives the meeting result for acceptation. When he validates the meeting solution, the application looks for the *Current* object in MaaoOTS library to commit the transaction. Being configured as “not null transaction context”, MaaoOTS library contacts with *OTSAgent* to get the transaction context, imposes this transaction context to the *Current* object and returns its reference to the application. The application then commits the transaction by invoking the *commit()*operation on this *Current* object.

6. Conclusion and Further Prospects

Although our OTS mobility add-in feature was initially designed for a telecom service, it can be applied in other architectures which need to solve the problems raised by transactions in the context of mobility. By using mobile agent technology, the OTS mobility add-in feature becomes generic and portable on different terminal platforms.

The mechanisms which we use to dynamically install the transaction service library and to migrate the transaction library state can be applied whenever there is a need to migrate on-the-fly software state-full libraries.

From an integration point of view, this paper shows an example where facilities made available by the mobile agent technology allow to extend the capabilities of a legacy service, in our case, a CORBA Transaction Service, in particular for the purpose of mobility.

Since one of the main objectives for the next generation telecom architecture is to harmonize different existing telecom networks, the agent technology can be considered as an extremely social partner which can transparently federate different domains for the purpose of accomplishing a well defined task. In our proposition, the pro-active agents guarantee the migration and configuration of a middleware among heterogeneous domains. These domains applications are unaware of the agents work, which is confined to a well identified task at the system level. This of course implies that, once for all, the domains have agreed to host the agent platform for that specific function

For the purpose of service personalization by the users, the Virtual Home Environment Provider will have to control user profiles and preferences. By definition these profiles will be distributed in the VHE provider, in the Service Providers and in the terminals. Distributed management of user profiles updates will be a key problem. Our results prove that it is feasible to deploy dynamically a transaction service in order to guarantee atomicity and isolation of profiles updates.

Further investigations are planned with regards to a roaming user, which moves to a new *VHE Provider*, which may or not provide its own transaction service.

7. Acknowledgements

This work has been performed in the framework of the project IST VESPER, which is funded by the European Community. The Author would like to acknowledge the contributions of his colleagues from Intracom Hellenic Telecommunications and Electronics Industry S.A., National Technical University of Athens, Institut National de Recherche en Informatique et Automatique, IKV++ GmbH Informations und Kommunikationstechnologie, GMD - Forschungszentrum Informationstechnik GmbH, Fondazione Ugo Bordoni, Universita' Di Catania, Portugal Telecom Inovação, University of Surrey, Technical Research Centre of Finland and SIEMENS AG Osterreich.

8. Glossary

3GPP	Third Generation Partnership Project
CORBA	Common Object Request Broker Architecture
EDGE	Enhanced Data rates for GSM Evolution
OTS	Object Transaction Service
OMG	Object Management Group
GSM	Global System for Mobile Telecommunication
GPRS	General Packet Radio Service
ISDN	Integrated Services Digital Network
PSTN	Public Switching Telephone Network
UMTS	Universal Mobile Telecommunication System
VASP	Value Added Service Provider
VESPER	Virtual Home Environment for Service Personalization and Roaming Users
VHE	Virtual Home Environment

References:

[1] 3GPP UMTS: Universal Mobile Telephone Service Specification - release 5, (http://www.3gpp.org/)

[2] TINA-C: Service Architecture, Architecture Specification: TINA-Consortium, 16 June, 1997.

[3] OMG OTS: CORBA Object Transaction Service, CORBA Services Specification: Object Management Group, Nov. 27, 2000.

[4] QIMING CHEN and UMESH DAYAL: Multi-Agent Cooperative Transactions for E-Commerce, *Cooperative Information Systems (CoopIS'2000)*, 2000

[5] M. DUNHAM, A. HELAL and S. BALAKRISHNAN: A Mobile Transaction Model that Captures Both the Data and Movement Behavior, *ACM-Baltzer Jounal on Mobile Networks and Applications (MONET)*, Vol. 2, No. 2 (1997). page: 149-62

[6] IST: Virtual Home Environment for Service Personalization and Roaming Users (VESPER), (http://vesper.intranet.gr/)

[7] JIAN LIANG, SIMONE SEDILLOT and BRUNO TRAVERSON: OMG Object Transaction Service based on an X/Open and ISO OSI TP Kernel, *Proceedings of the 30th Hawaii International Conference on System Sciences (HICSS-30)*, 1997

[8] IKV++ GRASSHOPPER: Grasshopper - Java Mobile Agent platform, (http://www.grasshopper.de)

[9] 3GPP VHE: The Virtual Home Environment, TS 22.121, Mar. 2001.

[10] VESPER: D22 - VHE Architecture design, 6/4/2001.

[11] R. KAROUI, M.SAHEB and S.SEDILLOT: Open Nested Transaction: A support for Increasing performance for Multi-tier Applications, *Transactions and Database Dynamics - TDD' 99*, 1999

[12] DANNY B. LANGE and MITSURU OSHIMA: Seven Good Reasons for Mobile Agents, *COMMUNICATIONS OF THE ACM*, Vol. 42, No. 3 (1999). page: 88-89

[13] PHAM HUY HOANG and SIMONE SEDILLOT: MaaoOTS Library Implementation Guide, Technical Report: TransRep Project - INRIA, Nov. 1999.