TECHNISCHE UNIVERSITÄT WIEN

VIENNA UNIVERSITY OF TECHNOLOGY

DIPLOMARBEIT

Shared Virtual Space Distribution Manager – SVSDM –

Use Cases

ausgeführt am Institut für

Computersprachen E 185

Programmiersprachen und Übersetzer E 185-1

unter Anleitung von

Ao. Univ. Prof. Dipl.-Ing. Dr. eva Kühn

eingereicht

an der Technischen Universität Wien

Fakultät für Informatik

von

marcus@mor.at

Wiedner Hauptstraße 18/1.Mezz/6

A-1040 Wien

Matrikelnummer: 9825311

Wieden, 18. Oktober 2005

TECHNISCHE UNIVERSITÄT WIEN

VIENNA UNIVERSITY OF TECHNOLOGY

DIPLOMA THESIS

Shared Virtual Space Distribution Manager – SVSDM –

Use Cases

conceived at the Institute of

Computer Languages E 185

Compilers and Languages Group E 185-1

supervised by

Ao. Univ. Prof. Dipl.-Ing. Dr. eva Kühn

submitted

at the Technical University of Vienna

Faculty of Computer Science

by

marcus@mor.at

Wiedner Hauptstraße 18/1.Mezz/6

A-1040 Vienna Austria

Registration Number: 9825311

Wieden, 18. Oktober 2005

in memoriam
Josefa Maier

Diese Arbeit ist bewußt in der alten deutschen Rechtschreibung gehalten,
da viele der neuen Regeln Sinnwidrigkeiten aufweisen
und sich damit gegen die deutsche Sprache selbst richten.

Kurzfassung

Viele Arbeitsvorgänge in Wirtschaft und Alltag verlangen die Verteilung von Daten. Viele Arbeiten werden heutzutage auf tragbaren Geräten (*mobile devices*) verrichtet. Deswegen wird die Verteilung von Daten immer wichtiger. Große Software-Firmen versuchen diesen Bedarf durch die Entwicklung von Synchronisationsprogrammen zu beheben. Diese sind oft sehr mächtig und benötigen daher viel an Computerleistung. Bei der Entwicklung solcher Programme wurde offenbar nicht beachtet, daß sie auf leistungsschwachen Rechnern (*thin clients*) zum Einsatz kommen.

Eine europäische Versicherungsgesellschaft hatte Bedarf an einer derartigen Software, die im Hintergrund, ohne viel Leistung zu beanspruchen, den Austausch von Daten gewährleisten soll.

Diese Arbeit gibt einen Überblick über die verwendete Hintergrundsoftware und ihre Anwendung in mehreren Einsatzszenarien. Am Beispiel einer Lösung für eine Versicherung mit mehreren Agenturen wurde ein Prototyp ausgearbeitet, als weitere Beispiele seien medizinische Notfallteams oder der Nutzen für Mitarbeiter eines Telephonnetzbetreibers genannt.

Abstract

Many tasks are asking for distribution of data. A lot of workers are already using *mobile devices* nowadays. Therefore distributing data is rapidly increasing in importance. Large software companies are trying to solve this issue by developing synchronizing tools. Most of them are complex and need quite a lot of computing power. They do not consider that such a tool should also work on a *thin client*.

A European insurance company experienced the need for a small piece of software to solve the synchronizing task in the background without consuming too much system performance.

This thesis gives an overview of the background software developed, demonstrating how it can be used in different appliances, for example by an insurance company with different agencies, as well by a mobile medical service, and a solution for the workers of a telecom service provider.

Acknowledgements

To say Thank You is something very personally, that is why I will write the Acknowledgements in my mother tongue.

I want to thank everyone who helped me realizing this diploma thesis !

Zu allererst möchte ich meiner Betreuerin Prof. eva Kühn danken, daß sie es mir ermöglicht hat diese Arbeit in die Wirklichkeit umzusetzen. Ich werde nie die drei Tage und Nächte im Jänner vergessen, wo wir gemeinsam mit Rania Khalaf den Proof of Concept für den SVSDM programmiert haben.

Weiters möchte ich Dipl.Ing. Roland Lutz erwähnen, der mit seiner großen Erfahrung aus Wirtschaft und Technik, oft mit seinen Ideen, große Probleme einer Lösung näher gebracht hat und der den Kontakt zu einer Deutschen Versicherung vermittelt hat.

Meine Hochachtung ergeht an Dipl.Ing. Richard Mordinyi, der die Vorarbeiten mit seiner Diplomarbeit geleistet hat, damit meine Arbeit überhaupt möglich wurde und der immer als ein besonderer Freund mir zur Seite steht.

Nun möchte ich noch alle Korrekturleser erwähnen. Allen voran meine Freundin Dina Dostal, die mich mit ihren Hinweisen vor allem auf Gedankensprünge aufmerksam gemacht hat. Dipl.Ing. Leopold Koppensteiner und Dipl.Ing. Karl Schiftner haben auch einige Korrekturen beigesteuert. Nicht zu vergessen ist auch eine gute Bekannte Dr. Ursula Kluwick, die vor allem der englischen Grammatik den letzten Schliff verpaßt hat.

Zum Abschluß möchte ich meinen Eltern danken, die mir vor allem in den letzten Monaten viel Zeit und Unterstützung geschenkt haben um diese Arbeit fertig zu bringen. Weiters möchte ich auch meine Großeltern erwähnen, die das Ihrige beigetragen haben.

Ich bedanke mich von ganzem Herzen bei jedem, der zu dieser Arbeit beigetragen hat !

Contents

I. Introduction 8

I.1. Motivation 8
I.2. Distribution 9
I.3. World Wide Web 10
I.4. Peer-to-peer 11
I.5. Problem description 12
I.6. Outline 15

II. Technical background 16

II.1. Web Services 16
II.2. BPEL 17
II.3. Middleware 22
II.4. Shared Virtual Space Distribution Manager 24
II.4.1. Overview 24
II.4.2. Inside view 27
II.4.3. SVSDM API 29
II.5. A classical solution 38

III. Use Case for an Insurance Company 40

III.1. Overview 40
III.2. Situation 41
III.2.1. Actual State 41
III.2.2. Problems 43
III.2.3. Supported Fields and Persons 44
III.2.4. Scenario 44
III.3. Prototype 46
III.3.1. Requirements 46
III.3.2. Solution 49
III.3.3. User Interface and Operation 51
III.3.4. Demonstration 62
III.3.5. Profiles 63

IV. Use Case for a Mobile Medical Service 64

IV.1. Situation 64
IV.1.1. Actual State 64
IV.1.2. Problems 65
IV.1.3. Requirements 66
IV.2. Possible Solution 68
IV.2.1. Workflow Management System 68
IV.2.2. Operation 70

V. Other Use Cases 72

V.1. Use Case for a Telecom Service Provider 72
V.1.1. Situation 72
V.1.2. Possible solution 72
V.2. Generic Use Case 74

VI. Evaluation 75

VI.1. Advantages 75
VI.2. Future Work and Improvements 77

VII. Conclusion 78

Acronyms 80

References 82

Printed material 82
Online material 84

Lists 85

Figures 85
Tables 86
Examples 86

Appendix 87

Integration of BPEL4J into an automated environment 87
Mobile Computing solutions by Xybernaut 90
Xybernaut 90
Wearable computing unit MA TC 90
Wireless display unit Atigo T 91
Modifications 93

I. Introduction

I.1. Motivation

The space based computing paradigm is a very new field of research where still many tasks are to be solved or many problems need a solution. In a lecture called "Middleware Programming" the first contact happened with the concepts of middleware systems. Target of the lecture was to encourage students in teams of two to develop programs to a given problem task description with the help of the middleware product called CoORdinated Shared Object (CORSO) [Kühn01] (based on the space based computing paradigm).

At that time a colleague and I worked together on a middleware problem description. He then created a generalized parcel service as diploma thesis [Mord05]. This service plays an important role in developing the solutions of the use cases described in this work.

One Use Case was developed together with a large German insurance company (AMB). I had the chance to demonstrate the prototype to their department of Information Technology (IT). They were very interested in my solution and are currently testing how it can be used within their company's IT infrastructure. Should the tests be successful they will probably start another project developing a real life product – which was the aim of this work.

This experience was the driving factor, to develop real life solutions that would make work tasks simpler. The creation of software together with users is a very exciting task. It requires having direct input of their needs and receiving feedback during the development process.

A major problem of software developers is the presumption that what *is believed* by the developer to be good for someone, has to be good for the user. Thus it happens in projects that functions are programmed which are believed to be needed but are actually not important for the user, while others that would be important are never realized at all.

The software described in this thesis was developed together with the users or at least in a field where the developer had knowledge of the context as a user himself.

Agile software development does not take any presumptions and is getting very important nowadays [www11]. The term agile software development came into common knowledge in 2001 when the initiators created their manifesto which includes four simple rules:

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

This type of software development is mend for teams (smaller than ten developers), which are confronted with unpredictable or rapidly changing requirements. These rules make it possible that user feedback can be included into the development as soon as they are mentioned.

At the beginning stands a short introduction to the "evolution of distribution".

I.2. Distribution

The WikipeadiA[1], the free (online) Encyclopaedia tells the following for the term "distribution" [www01]. **Distribution** can mean:

- In mathematics, there are several distinct concepts given the name of *distribution:*
  - Generalized functions.

[1] **WikipediA** is a Web-based, free-content encyclopedia written collaboratively by volunteers and sponsored by the non-profit Wikimedia Foundation. [www01]

    - Probability distribution.
    - Carnot-Cartheodory manifolds, sub-Riemannian manifold.
- In physics, a *distribution* function, for example the Maxwell-Boltzmann distribution, describes the number of particles per unit volume in phase space.
- In Business operations, *distribution* is one of the four aspects of marketing.
- For computing science concept, distributed computing.
- For the meaning of *distribution* in the terminology of the Linux operating system.
- Electricity distribution.

This thesis wants to concentrate on the terms "distributed computing" and "distributed systems". A.S. Tanenbaum and M.v. Steen [Tane02] give the following definition:

> A distributed system is a collection of independent computers that appears to its users as a single coherent system.

This statement addresses two aspects: On the one hand it talks about hardware. On the other hand it refers to a particular kind of software. This software has to create the assumption of "a single coherent system" [Tane02].

The focus of this thesis is to create this vision and to care for the software issues. The Appendix contains a detailed description of the mobile hardware that was utilized in the presented use cases.

I.3. World Wide Web

Distribution (of data, information) started with the introduction of the Web[2].

The web was created 1989 as a project at the CERN[3] in Geneva (Switzerland) by Tim Berners-Lee. The challenge was to find an easier way to exchange research results with colleagues.

---

[2] The **World Wide Web ("WWW", "W3", or simply "Web")** is an information space in which the items of interest, referred to as resources, are identified by global identifiers called Uniform Resource Identifiers (URI). The term is often mistakenly used as a synonym for the Internet, but the Web is actually a service that operates *over* the Internet. [Rech99]

The http[4] together with easy-to-use Web browsers achieved a public breakthrough. Since then everyone has been able to gain easy access to a huge information source, which nowadays enables the distribution of information through the use of Web browsers. The Web has not remained without its critics, however, who take issue with the fact that information circulated in that way tends not to be prechecked for accuracy or is even incorrect.

Today the World Wide Web can be seen as a distributed system. When browsing through web pages, they have been assembled through a distributed system, allocated around the world and consisting of different servers.

Servers are still needed to hold the information until a user picks it up from there. For example in order to use the mail system the POP3[5] or IMAP[6] Server has to be known, which holds the mailbox.

I.4. Peer-to-peer

A different attempt is the so called Peer-to-Peer (P2P) approach [Dust03], which is used to exchange data (*file sharing*) or for instant messaging, where messages are exchanged (*chatting*). P2P uses a network of computers with the endpoints having equal rights. Both communication partners are on the same step of complexity [Mina01] with no distinction between *client* (a piece of software accessing a server) and *server* (a centralized service provider).

---

[3] European Organization for Nuclear Research, French: Organisation Européenne pour la Recherche Nucléaire previously called **Conseil Européen pour la Recherche Nucléaire (CERN)**. [Rech99]

[4] **HyperText Transfer Protocol (HTTP)** is the primary method used to convey information on the World Wide Web. The original purpose was to provide a way to publish and receive HTML pages (HyperText Markup Language [a language for creating web pages]). [Rech99]

[5] **Post Office Protocol version 3 (POP3)** is an application layer Internet standard protocol used to retrieve email from a remote server to a local client over a TCP/IP (Transmission Control Protocol/Internet Protocol is the name for the Internet protocol suite) connection. [Rech99]

[6] The **Internet Message Access Protocol** (commonly known as **IMAP**, and previously called Interactive Mail Access Protocol) is an application layer Internet protocol used for accessing email on a remote server from a local client. [Rech99]

The term P2P came into common knowledge in 1999 with the creation of the program Napster by Shawn Fanning. It was intended as an easy to use search engine for finding shared mp3[7] files. Each individual *user* had to download a client, which then was installed on the own computer. Each *client* had to contact a central server, where all the files to be shared where registered. Napster provided a self-explanatory, user friendly GUI[8], where users could search for music files listed at the central server. The actual download was done directly from the provider of the searched file. The P2P model used is the so called data centred model, where a central server is used to index the distributed data. Different models can be identified. See the following documents for more information [Kanh02][Riem03][Gart01].

The product described above worked world wide with one single server as - because of the design of the clients - the biggest load (the downloading of the data) was handled between the two clients (searcher to provider, *peer-to-peer*). Napster showed that simple software can create a huge success. In 2001 this program was shut down after many lawsuits. The central server, where the data about the music files were stored, proved to be the legal problem and so copyright issues became suable.

After that some more P2P applications were created very quickly (Gnutella, FastTrack, P-Grid [Dust03]), which worked fully decentralised. In order to communicate with all of them, only one member has to be known. Creating fully self-controlling software, without the need of a central coordinator is a very new point of view.

This sounds easy and natural, but in reality that may look different.

I.5. Problem description

In today's networks often complex organisational structures are represented, where central computers, so-called *servers*, control all the flow of information. There exists a centralized service provider, where the business logic is implemented and the database is situated.

---

[7] **MP3 (MPEG-1 Audio Layer 3)** is a popular digital audio encoding and lossy compression format. It was designed to greatly reduce the amount of data in audiofiles. [www01]

[8] A **Graphical User Interface (GUI)** is a method of interacting with a computer through a metaphor of direct manipulation of graphical images and widgets in addition to text. [www01]

All participants, or so-called *clients*, have to connect to one (or more) servers if they want to use that network. Each client only knows the servers and it is not possible for it to contact another client without the overhead of using a server, well known to both clients. They are needed to query or use the above mentioned services.

Those networks are dependent on servers and are called *Client/Server Networks* [Orfa99] . A good example of that type of network is the Internet, where all information is stored on servers (web-, email-, newsgroup-server, etc.). Users may collect the information through different clients.

The solution described above is the classic one.

Nowadays, with the introduction of a great number of mobile devices, the term distribution is getting far more important. Nearly everyone has a mobile device and wants to be able to access her/his data or services all the time from different places. To achieve that two solutions exist: Either being online all the time (this proves to be very expensive needing to have a mobile connection) - or having to synchronize all needed data and services with the mobile device. This transforms it actually into a so called "*fat client*", a copy of the centralized service provider with a sub selection of the data. Due to the data storage components needed, such a client is similarly expensive. Furthermore, it is not a good solution to supply all users with a copy of the service provider, which was intended to be kept centralized to facilitate the updating and maintaining of its functions.

In view of this problem it is evident that the need of component based software development has become very important. A standard way of doing so is by the usage of *web services*.

> Web Services are a new breed of **Web applications**. They are **self-contained, self-describing, modular** applications that can be **published, located,** and **invoked across the Web. Web services perform functions,** which can be anything from simple request to complicated business processes... Once a Web service is deployed, other applications (and other Web services) can discover and invoke the deployed service. [www09]

This statement by IBM tells the key features of web services. This sounds very useful but it requires a big overhead of providing all the features described above. The partial

function needed for the location service (so called UDDI[9] Service) is still not as useful as wished. Furthermore, while using a service supplied by a different provider, staying online is required that is still quite expensive. When needing many different service providers these online calls are time-consuming tasks and often slow-down a software solution. A solution which takes into account the problems arising when being offline would be better and faster. (For more detailed information see Chapter II.1)

However an out-of-the-box solution will prove disappointing since they are difficult to find in a shop. Even then they often have to be customized very complicated for the operation. That is why a large German Insurance Company was not satisfied and a new solution had to be developed. They asked for software supporting their mobile agents while visiting clients. Key requirement was that a pool of tasks should be automatically transmitted to the first agent available and capable of fulfilling the order. In the previously used system a task could only be forwarded to one specific client, not to a client group with specific requirements. A second demand was that tasks should be retransferrable to another agent if the original task holder could not finish it within a given time span. (For further technical details see Chapter II)

The solution should support updating the stored data of the mobile device, without needing any active start action. So as soon as an online connection is available new packets destined for that agent are automatically pushed up. This is achieved by using the background software called SVSDM[10] (for more information see Chapter II.4 Shared Virtual Space Distribution Manager) [Mord05].

SVSDM solves most of the distribution issues. It is a P2P system that in addition uses some of the advantages of a central server to coordinate the stream of data.

The challenge was to find a solution compatible with the security scheme of a large company and at the same time flexible enough to serve mobile agents. Let us look at a small example to ease understanding:

---

[9] **Universal Description, Discovery, and Integration (UDDI)** is a platform-independent, XML-based registry for businesses worldwide to list themselves and their services (web services) on the Internet. [Dust03]

[10] **Shared Virtual Space Distribution Manager (SVSDM)** is a software creating a Virtual Space shared between different clients and managing the distribution of the data objects. [Mord05]

Think about a supplier of beverages. The beverages are transported on big vans and each driver has a route to follow. This information is stored on a mobile device, which was synchronized at the starting point of the tour. It is important that the data is not only stored on that device (through a *message passing system*) as it can happen that this device may break down or be stolen. If SVSDM is used, the data that was copied onto that device are marked as in use by a client on the information provider. When a different client tries to fetch this information during an authentication with the same IDentification (ID), the system will grant all rights to this data to her/him. There also may be a defined timeout. After its expiration all other users can get access to that data again.

I.6. Outline

Chapter II Technical background gives an introduction to web services, to the workflow topic, to SVSDM and why middleware is very important nowadays.

Chapter III Use Case for an Insurance Company, Chapter IV Use Case for a Mobile Medical Service, and Chapter V Other Use Cases are the main parts of this thesis and describe the use cases starting from the description of the basic situation, the requirements, leading to the solution of the problem by using SVSDM, and showing the benefits (system independence, recoverability, short development times, less source code, transactions, minimal network traffic, offline mode, monitoring) that arise through that application.

Chapter VI Evaluation will assess the outcome and give some future aspects.

Chapter VII Conclusion gives a summary of the work.

II. Technical background

This chapter gives highlights the technical background. The technology important to this thesis is presented and its features and alternative solutions are subsumed.

First Web Services are described, as they are a very new development making module-based software development possible. Afterwards a Business Process Execution Language (BPEL) is introduced. This language is a workflow description language using the Web Service approach. Finally an introduction to middleware systems is given, starting with the historical development and describing some important featured provided. The chapter finishes with a very detailed essay on the SVSDM, a Virtual Space Manager (VSM) based distribution manager.

II.1. Web Services

A web service is a new kind of web application [Dust03]. The idea behind it is easy to understand. A web service is part of a full system, which is split into modules. Each module may be deployed on a different computer. The full function is available when many computers collaborate. The key idea of packing is that special services may be supplied by different providers, or, alternatively, that one service may be used by different requestors. The services may also be able to search for necessary functions over the internet. Therefore a so called UDDI – Service was conceived which stores the WSDL[11] File - a description of the Web Service in a special language. It defines the

[11] **Web Services Description Language (WSDL)** is an XML[13] format published for describing Web services. [Dust03]

layout of the messages that are exchanged through SOAP[12] over the Internet. SOAP defines an XML[13] messaging protocol. The UDDI – Service is used to make an enquiry for a special service. If an answer is given the web service will contact the service found in this way that will fulfill the needs.

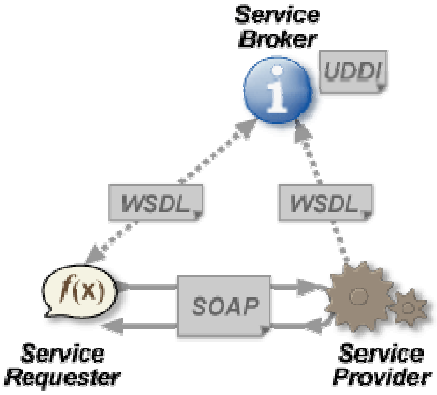

Figure 1: Web Service [www01]

For Example: The task is the planning of a business travel to London and therefore to create a web service that should find the cheapest way to get and stay there. The web service will first try to find out a route of how to go there contacting different airlines, bus and perhaps also a train services. The web service may first search for all these travel facilities and then will try to find out the price for the tour. Then it will search for a hotel. For this part some preferences might be specified, such as that the place should be near to the conference hall. The web service will automatically store all routes found and evaluate the prices. The cheapest and best fitting route will then be presented to the user.

II.2. BPEL

Business Process Execution Language (BPEL) [Andr03][Dust03][Juri05][www06] is an extension to web services. This language is XML based and was created 2003 by a cooperation of big world wide acting companies (IBM, Microsoft, BEA-Systems,

[12] **Simple Object Access Protocol (SOAP)** is a standard for exchanging XML[13]-based messages over a computer network, normally using HTTP. [Dust03]

[13] The **eXtensible Markup Language (XML)** is a simplified subset of SGML (Standard Generalized Markup Language). XML enables authors to define their own tags. XML is a formal specification of the World Wide Web Consortium. [Stuc05]

SAP AG, Siebel Systems and others). The intention behind creating such a language was to find a modern business process description language. The concept of this language is mainly a unification of the WSFL (the Flow Language of IBM) that is based on the concept of direct graphs and XLANG (by Microsoft), a block-structured language. BPEL is a combination of these two languages and offers a rich vocabulary for the depiction of business processes. BPEL can be used as a description language of a business flow (*abstract business protocol*) as well as a programming language for web services (*executable process*).

With BPEL a large number of web services can be organized to cooperate and follow a greater target. The idea is to split the work task into small parts (services) which are implemented independently. Later they are put together with the help of a workflow. This makes it possible that frequently used services (for example: from different workflows or different companies) may be provided by one single service provider. A web service may even be a manual task, for example filling in a form or asking a worker to question the client. BPEL allows describing a very complicated business case and enables programming in the large (that complicated tasks being split into modules).

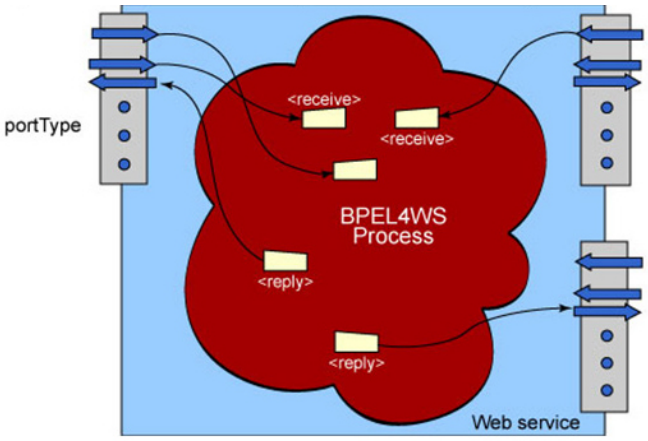

Figure 2: BPEL Process [www06]

The BPEL process (see Figure 2, the red cloud) is a flow-chart-like expression of an algorithm. Each step is called an *activity*. See Table 1 for a collection of *primitive* activities.

These activities can be combined to a more complex algorithm using any of the provided *structure* activities (see Table 2).

Tag	Meaning
<invoke>	invoking an operation on some web service
<receive>	receiving an invocation by someone externally
<reply>	replying to an input/output operation (synchronous)
<wait>	waiting for some time
<assign>	assigning a value or copy data to a variable
<throw>	throwing an error, indicating that something went wrong
<terminate>	terminating the entire service instance
<empty>	doing nothing

Table 1: BPEL primitive activities

Tag	Meaning
<sequence>	defining an ordered sequence of steps
<flow>	indicating that a collection of steps may be executed in parallel
<switch>	case-switching for implementing branches
<while>	defining a loop
<pick>	executing one of several alternative paths

Table 2: BPEL structure activities

Each BPEL process has to define partner links (<partnerLink>) and to declare variables (<variable>). A partner link specifies a partner that interacts with the BPEL process. Each partner link has a specific `partnerLinkType` that characterizes it and one ore two of the following attributes:

- `myRole`: the role of the business process
- `partnerRole`: the role of the partner

Variables in BPEL processes are used to store, reformat, and transform messages. Usually there will be a variable for every message sent to or received from partners.

```
<process name="BusinessTravelProcess" ... >

   <partnerLinks>
      <!-- The declaration of partner links -->
   </partnerLinks>
   <variables>
      <!-- The declaration of variables -->
   </variables>
   <sequence>
      <!-- The definition of the BPEL process body -->
   </sequence>
</process>
```

Example 1: Example for an empty BPEL Process

For an easier understanding an oversimplified business process [www10] for employees travel arrangements is shown that illustrates which activity is used. This can be referred to as the BPEL process body (see Example 1). The client (employee) invokes the business process (see Example 2 for a more detailed example), specifying the name of the employee, the destination, the departure date, and the return date. Action <receive> will wait for that invocation. Then the business process will be instantiated and start gathering the travel information. First it will check the employee's travel status (to be able to do that an <assign> action has to be done, copying the employee's name to the message that will be passed on). Assuming that there exists a web service by which such a check can be done, action <invoke> will do that. Here it is possible to specify an input and output variable using the synchronous invoke activity, as receiving this information might be quick. After that there will be another <assign> activity to generate the message. This will later be passed on to two further <invoke> activities. These will ask the web service of two different airlines for their flight plans. These two activities may be started in parallel (using a <flow> activity). Here the asynchronous invoke activity is used and therefore a <receive> activity has to be put after each invoke, which will wait for the answer, as the flight companies might need some time for that. After all the information has been collected a <switch> activity will decide to return (using <assign> activities) the cheapest flight offer using an <invoke> back to the client.

In this example four <partnerLink> elements can be identified. The first one is the employee, the client. The second is the check for the travel status and the last two are airline services.

There is the need for seven <variable> elements, each storing a message, either to send to or to receive from a partner.

```
<process name="BusinessTravelProcess" [...] >
   <!-- The declaration of partner links -->
   <partnerLinks>
     <partnerLink name="client"
                  partnerLinkType="trv:travelLT"
                  myRole="travelService"
                  partnerRole="travelServiceCustomer" />
     [...three more partnerLink definitions...]
   </partnerLinks>
   <!-- The declaration of variables -->
   <variables>
      <!-- input for this process -->
      <variable name="TravelRequest"
                messageType="trv:TravelRequestMessage" />
      [... six more variable definitions ...]
   </variables>
   <!-- The definition of the BPEL process body -->
   <sequence>
      <!-- Receive the initial request for business travel from client -->
      <receive partnerLink="client"
               portType="trv:TravelApprovalPT"
               operation="TravelApproval"
               variable="TravelRequest"
               createInstance="yes" />
      <!-- Prepare the input for the Employee Travel Status Web Service -->
      <assign>
        <copy>
          <from variable="TravelRequest" part="employee" />
          <to variable="EmployeeTravelStatusRequest" part="employee" />
        </copy>
      </assign>
      <!-- Synchronously invoke the Employee Travel Status Web Service -->
      <invoke partnerLink="employeeTravelStatus"
              portType="emp:EmployeeTravelStatusPT"
              operation="EmployeeTravelStatus"
              inputVariable="EmployeeTravelStatusRequest"
              outputVariable="EmployeeTravelStatusResponse" />
      <!-- Prepare the input for A1 and A2 -->
      [...like the preparation for the input for the Employee Travel Status...]
      <!-- Make a concurrent invocation to A1 in A2 -->
      <flow>
        <sequence>
          <!-- Async invoke of the A1 Web service and wait for the callback-->
          <invoke partnerLink="Airline1"
                  portType="aln:FlightAvailabilityPT"
                  operation="FlightAvailability"
                  inputVariable="FlightDetails" />
          <receive partnerLink="Airline1"
                 portType="aln:FlightCallbackPT"
                 operation="FlightTicketCallback"
                 variable="FlightResponseA1" />
        </sequence>
        <sequence>
          <!-- Async invoke of the A2 Web service and wait for the callback-->
          [...same as above...]
        </sequence>
      </flow>
      <!-- Select the best offer and construct the TravelResponse -->
      <switch>
        <case condition="bpws:getVariableData('FlightResponseA1',
                        'confirmationData','/confirmationData/Price')
                        <= bpws:getVariableData('FlightResponseA2',
                        'confirmationData','/confirmationData/Price')">
          <!-- Select Airline1 -->
          <assign>
            <copy>
              <from variable="FlightResponseA1" />
              <to variable="TravelResponse" />
            </copy>
          </assign>
        </case>
        <otherwise>
          <!-- Select Airline2 -->
          [...same as above...]
        </otherwise>
      </switch>
      <!-- Make a callback to the client -->
      <invoke partnerLink="client"
              portType="trv:ClientCallbackPT"
              operation="ClientCallback"
              inputVariable="TravelResponse" />
   </sequence>
</process>
```

Example 2: Example for a BPEL Process (exzerpts)

Above the definition of a workflow was described using BPEL – a workflow description language. Each activity in the workflow has to fulfill a predefined task. This may be implemented in any programming language using the predefined interfaces (BPEL defines them). These tasks may be anything from a simple lookup in a database to a difficult compilation of steps using a complicated logic. To achieve that goal the usage of middleware software may be suggestive.

II.3. Middleware

Middleware is an infrastructure situated on a layer between the application and the system software and the network layer [Mahm04][Call97]. It gives the developer an abstracted view of the underlying layers and so offers the possibility for the application to run on all systems that are supported by that middleware. One definition of middleware says that:

> `middleware is software sold to people who don't know how to program by people who know how to program` [www03].

This definition may be interpreted in the following way: Middleware creates the possibility to reuse parts of software, where difficult and very sophisticated solutions (for example: very expensive technology for supporting online/offline situations) are implemented only once and probably by a special organized team. Therefore an application may be developed much faster as many time-consuming implementations of fundamental functions are provided [www08]. The created program will run on many different operating systems as the middleware provides the same interface on all different supported systems. To achieve that, parts of the middleware have to be developed specially on each system to support different system specific protocols. The middleware creates a defined platform on different systems, which acts as a basis on which application software may be developed. This software will then be available on all systems, which are supported by the middleware used [Kühn98].

Middleware technologies have been developed and successfully introduced into fixed networks [Vino04]. There they create a distribution *transparent* to both the user and the software engineer, so that systems appear as a single computing facility [Sutt01]. However, completely hiding implementation details from the application makes it more

difficult and often creates obstacles in a mobile setting as mobile systems need to react quickly to changes happen in their environment.

To overcome this lack of mobility this work is based on the SVSDM, which was created at the Technical University of Vienna using a technology developed during the last 15 years. In 2004 the produced work depending on this technology was subsumed under the SBC-Grid[14] initiative. They define SBC-Grid software architecture (see Figure 3) to consist of the following layers (see Table 3).

Layer	Purpose
Products or Applications layer	Products and applications that make use of the other layers.
Patterns layer	Open source infrastructure layer that implements reusable software coordination design patterns.
Service layer	Open source infrastructure layer that contributes to the self-description and self-organization of the software architecture.
Kernel layer	Middleware layer that is based on the space based computing paradigm, like e.g. CORSO [Kühn94], eXtensible Virtual Shared Memory (XVSM) [Kühn01], or JavaSpaces [Free99].

Table 3: Layers of the SBC-Grid Architecture

[14] **Space Based Computing - Grid (SBC-Grid)** see [www07] for more information.

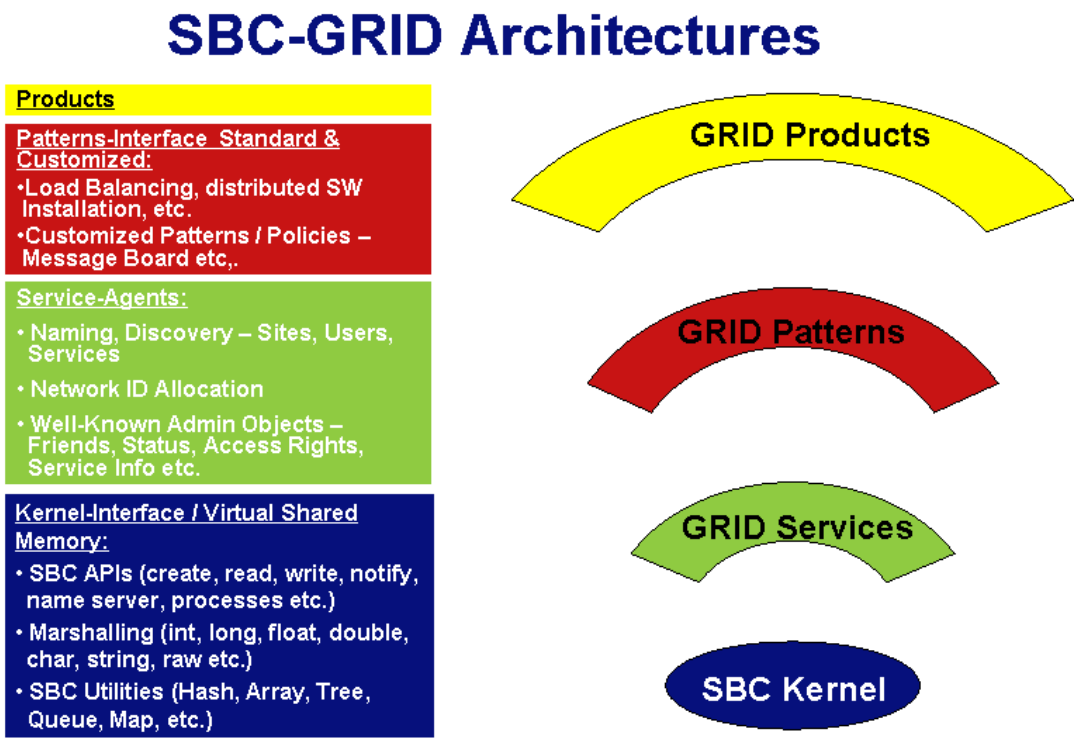

Figure 3: SBC-GRID Architectures [www07]

The SVSDM (see Chapter II.4 for more information on that topic) can be classified as a pattern as it provides reusable software for utilisation in different use cases.

II.4. Shared Virtual Space Distribution Manager

II.4.1. Overview

Most of the functions were developed based on the requirements of the first use case (see Chapter III). Some more were added to make the solution universally valid, and to enable other applications as well. The different use cases will show how to use these functions in different scenarios. The functions listed (see Table 4) are provided by the SVSDM. They will be described in more detail in the following paragraphs.

The first step in the implementation is to distribute data; therefore one needs an *import* and an *export* functionality. These two functions are the interface to the software that was developed on top of the SVSDM. Looking at SVSDM this way one can call it middleware software, because it manages the network traffic and coordinating the communication for the overlying application software. SVSDM was implemented using Java, which makes it more easily portable to different platforms, as a Java

implementation exists for nearly all available platforms. It also supports some features, as *pack* and *unpack* and automatically *start* a system call on the destination system. In addition, the possibility exists to *return* the answer from that call and *return* it to the caller. Another function that was built into SVSDM is the *Monitor/Display*. This provides a way of depicting what happens within the system. Every function call is logged. Therefore it is possible to track every work package from the producer to the consumer and if an answer is returned, even that. This helps the developer, while she/he implements the application, since it highlights possible errors in the real life situation and helps her/him trace what was done by whom.

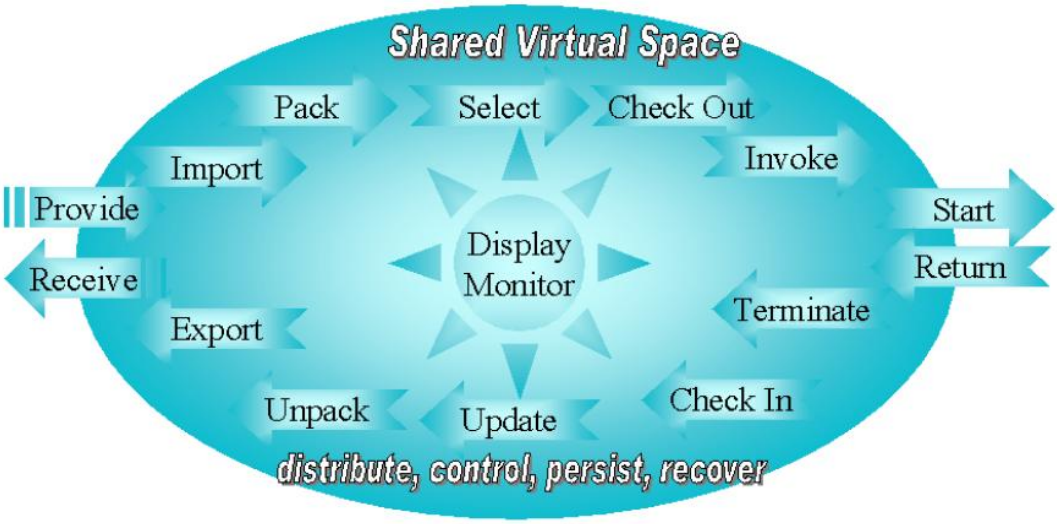

Figure 4: SVSDM Functions

Function	Managed by	Description
Provide	User SVSDM	providing data, that has to be distributed
Import	User/Client SVSDM	importing data into the SVSDM
Pack	User/Client SVSDM	packing data into the SVSDM Packet
Distribute	SVSDM	distributing of the Packets with the help of user specific Profiles if available
Check Out	SVSDM	checking out selected Packages onto the target system
Invoke	User/Client SVSDM	preparing (invoking) the target application with the contents of the retrieved packet
Start	User	executing (starting) the application
Return	User/Client SVSDM	returning potential results
Check In	User/Client SVSDM	checking the Packet into the system with the newly added contents
Collect	SVSDM	collecting answer Packets
Unpack	User/Client SVSDM	unpacking data from the SVSDM Packet
Export	User/Client SVSDM	exporting answer to the calling system
Receive	User	receiving returned results
Monitor / Display	User SVSDM	logging (monitoring, displaying) usage of the system

Table 4: List of Functions provided by SVSDM

II.4.2. Inside view

SVSDM is a pattern using a Shared Virtual Space that is a simulated joint memory that can mask network failures and handle offline situations.

An offline situation cannot be masked completely and therefore the software will inform the application on top of it, that the online connection is not available at the given moment. To manage those situations more convenient a Global Persistent Temporary Space (GPTS) and a Local Persistent Temporary Space (LPTS) are provided by SVSDM. As indicated by the names these spaces are thought to be temporary only. They are not implemented as persistent memory and not laid out for a huge amount of data. Temporary also means that the saved data is only needed during the active distribution process; afterwards all the data may be deleted, or stored outside of SVSDM. The GPTS is the global storage which will be widespread over all computers working together. The GPTS may be used by all the different clients to read, store and share information. The LPTS is to be seen as a very short time memory, which is used to save the data that was downloaded from the GPTS for offline usage or that will be uploaded the next time an online connection is available. The LPTS is the tool to keep data during the offline state.

Many of these exceptional situations are solved by the underlying software called CORSO (see [Mord05] for more information on that topic). As a very important factor of SVSDM, the client part is equipped with a functionality enabling it to find out whether it is connected to the global space or not. In the mobile computing scenario offline working is very important and this is the main difference to the old hard wired computing era. Therefore at any given time the client has to know whether a connection exists or not and should be able to report this circumstance to the user, reporting about the availability of the full functionality.

To make this possible, a function called `LifeBeatChecker` was implemented into the clients. Its function is very simple: A shared object stored at the GPTS will be connected with a notification to the client. It will periodically increment its value. This change will trigger the notification and the client will get informed of the operating online connection. Furthermore, if the new value is different from the situation before the change, the client also gets to know something about the functionality of the server. A change means that everything is alright.

The main task of the SVSDM is the distribution. Therefore be reminded of the simplified figure in the previous chapter (Figure 4: SVSDM Functions). The main functions are definitely import and export, in other words the creation of an object and its reading out. To explain how this works the life of an object will be shown: First the object has to be created `new Packet(content, sender, receiver)`. This object has to be inserted into the LPTS, that can be achieved with the call `lpts.addPacket(packet,ttl)`. The time-to-live[15] (TTL) tells the system how long the packet is supposed to be available in the space. Another function takes care of the timed out packages. This is called the `ThreadGarbageCollector`. This thread runs in the background and periodically checks the TTL of all packets and automatically removes timed out packages. This functionality is implemented into the GPTS beforehand but if needed at the LPTS it has to be started manually (This solution was chosen because the garbage collector is not always needed at the LPTS, e.g. at the producer side all packets are uploaded to the GPTS not caring for the TTL, at the consumer side this looks different, as the packets will be kept there until timed out.).

At the client as well as at the central unit it seems interesting to show the users the contents of the spaces, therefore a functionality implemented by the SVSDM is used. This table, the `NotificationBoard`, always lists the contents of the LPTS or GPTS, and if changes happen they are immediately shown on the table. This is possible through a complex notification service that will be described a little later. The same feature helps to keep the GPTS updated if there are any changes in the LPTSes and naturally also in the opposite direction. When creating the table a filter may be referred to (this may only be useful if used with the GPTS), in order to restrict the view of some of the contents.

SVSDM uses space based computing software (CORSO) that provides a very sophisticated transaction system. This system makes sure that functions, called in semantic coherence, will be done as a whole or kept back at all. Two different transactions are supported: The Top-Transaction is a data transfer that is not nested into another one. It starts a completely new transaction scope and is autonomous so that no other transaction is dependent on it. The second sort of transactions are the

---

[15] **time-to-live (TTL)** is a limit on the period of time that a unit of data (e.g. a record) can exist before it is discarded. [www01]

Sub-Transactions, which are either nested in a top level one or into an existing sub transaction.

The system also distinguishes between two types of commitments: There is a "hard" commitment, where the transaction is executed successfully or is aborted. When using the "soft" commitment the behaviour is the same in case of success, but in case of failure one is able to restart the failed sub transaction once again or react to an error message.

Another feature is the notification system supplied by CORSO. SVSDM completely omits the usage of polling (checking data or objects for change in a predefined timeout). Therefore notifications are used. This procedure is more efficient than polling because it minimises network traffic. It works quite simple: The developer creates a list of notification items, any shared object may be added. If the value of these objects changes, the notification list will fire and the program may react to that in a predefined way. For example the list is filled with different objects and in case one changes, the items fires. The execution will determinate which type of object was subject of change and invoke the correct process to handle it.

II.4.3. SVSDM API

This chapter will give an overview of the SVSDM API[16].

II.4.3.1. Class Global Persistent Temporary Space

The object `GlobalPTS` is created using the following call:

```
gpts = new GlobalPTS(connection,
                     name_of_packetdirectory,
                     name_of_communicationdirectory,
                     properties,
                     create_flag)
```

Example 3: Call to create new GPTS

[16] An **Application Programming Interface (API)** is a set of definitions of the ways one piece of computer software communicates with another. It is a method of achieving abstraction. [www01]

The submitted parameters represent the following:

- `connection:`
  is the object that encapsulates the link to the underlying CORSO software. This object specifies to which host the SVSDM should connect.
- `name_of_packetdirectory:`
  is a freely chosen name that will be given to the shared object, which will represent the packet directory of the GPTS, all packets will be saved here.
- `name_of_communicationdirectory:`
  is the name for an object representing the communication directory of the GPTS. This is used to manage the message interaction between GPTS and clients with a LPTS.
- `properties:`
  contains some special attributes for example the `TTL_GPTS` - which specifies the time-to-live of the objects stored in the GPTS.
- `create_flag:`
  this attribute controls the creation of the GPTS, if set to `true` the GPTS will be created when not existent, if set to `false` the call will fail if the space is not yet available at the host.

Furthermore the so created object `gpts` provides the following calls:

- `gpts.startServices():`
  starts the services provided by the GPTS, that would be the `life_beat`, `communication`, and `garbagecollector`.
- `gpts.stopServices():`
  stops the services mentioned above.
- `gpts.addPacket(packet, ttl):`
  adds a packet to the GPTS, the time-to-live has to be given.
- `gpts.deletePacket(oid, transaction)`, `gpts.deletePacket(oid):`
  deletes the packet specified by the Object ID (OID), either using a given transaction or implicitly.
- `gpts.showPacketDirectoryContentOnce(table_element)`, `gpts.showPacketDirectoryContentOnce(name_of_packetdirectory,`

`table_element)`,
`gpts.showPacketDirectoryContentOnce(name_of_packetdirectory, location_of_packetdirectory, table_element)`:
lists the contents of the GPTS or a different packet directory on the same/different host in a GUI table.

- `gpts.registerUser(id, name_of_communicationdirectory)`:
  registers a new user, identified by its ID and the name of the communication directory.

- `gpts.unregisterUser(id, name_of_communicationdirectory)`:
  removes the given user ID from the named communication directory.

- `gpts.listUsers(name_of_communicationdirectory, table_element)`, `gpts.listUsers(name_of_communicationdirectory):Hashtable`:
  is used to get the authenticated user list, either in a hash table or prepared to be filled into a GUI table.

- `gpts.resetComm()`:
  is used to actualize the internal list of authenticated users after new registration or deletion, will be avoided in a later version.

II.4.3.2. Class Local Persistent Temporary Space

The object `LocalPTS` is created using the following call:

```
lpts = new LocalPTS(connection,
                    name_of_lpts_packetdirectory,
                    name_of_gpts_packetdirectory,
                    name_of_communicationdirectory,
                    location_of_gpts,
                    create_flag,
                    user_id)
```

Example 4: Call to create new LPTS

The submitted parameters represent the following:

- `connection`:
  is the object that encapsulates the link to the underlying CORSO software. This object specifies which host will be used for the LPTS.

- `name_of_lpts_packetdirectory:`
  is a freely chosen name that will be given to the shared object, which will represent the packet directory of the LPTS. All local packets will be saved there when there is no online connection or the packet is in use.
- `name_of_gpts_packetdirectory:`
  is the packet directory name of the corresponding GPTS.
- `name_of_communicationdirectory:`
  is the name for the communication object enabling the message interaction between GPTS and LPTS.
- `location_of_gpts:`
  this is either an IP-address[17] or an identifier (used in systems, where the IP-address uses to change) for a computer which hosts the GPTS.
- `create_flag:`
  this attribute controls the creation of the LPTS, if set to `true` the LPTS will be created when not existent, if set to `false` the call will fail if the space is not yet available at the host.
- `user_id:`
  is the identifier for the user who will use this LPTS.

Furthermore, the object `lpts` so created provides the following calls:

- `lpts.checkMyCommunicationStatus():Boolean:`
  returns `true` if the LPTS that was created using a specific user ID is allowed to communicate with the GPTS.
- `lpts.GPTS2LPTS(oid, ttl):`
  transfers the object with the given ID to the LPTS setting the time-to-live to the passed value.
- `lpts.LPTS2GPTS(oid):`
  works the other way round; moves the object with the given ID to the GPTS.

---

[17] An **Internet Protocol address (IP address)** is a unique number, similar in concept to a telephone number, used by machines (usually computers) to refer to each other when sending information through the Internet. [Rech99]

- `lpts.placeInLPTS(packet, ttl):`
  places a packet with the time-to-live into the LPTS.

- `lpts.storeContent(oid, packet, transaction):Packet,`
  `lpts.storeContent(oid, packet):Packet:`
  stores the contents of the object with the given ID either using the given transaction or implicitly into a packet that will be returned.

- `lpts.showPacketDirectoryContentOnce(table_element),`
  `lpts.showPacketDirectoryContentOnce(name_of_packetdirectory, table_element),`
  `lpts.showPacketDirectoryContentOnce(name_of_packetdirectory, location_of_packetdirectory, table_element):`
  lists the contents of the LPTS or a different packet directory on the same/different host in a GUI table.

- `lpts.changePacketStatus(oid, status_code, transaction),`
  `lpts.changePacketStatus(oid, status_code):`
  changes the status code of the object identified by the ID either using the given transaction or implicitly.

- `lpts.deletePacket(oid, transaction),`
  `lpts.deletePacket(oid):`
  deletes the packet specified by the object ID, either using a given transaction or implicitly.

II.4.3.3. Class Notification Board

The object `NotificationBoard` is created using following call:

```
notifboard = new NotificationBoard(connection,
                                   restrictions,
                                   table_element,
                                   name_of_packetdirectory,
                                   location_of_packetdirectory,
                                   show_packets_or_users)
```

Example 5: Call to create new Notification Board

The submitted parameters represent the following:

- `connection:`
  encapsulates the link to the underlying CORSO software.

- `restrictions:`
  defines constraints on the elements that will be shown in the table.
- `table_element:`
  this is a GUI element representing the table, where elements will be shown.
- `name_of_packetdirectory:`
  is the packet directory name of the directory, whose contents will be shown on the table.
- `location_of_packetdirectory:`
  this is either an IP-address or an identifier (used in systems where the IP-address frequently changes) for a computer which hosts the packet directory.
- `show_packets_or_users:`
  is a switch that controls which kinds of elements are shown in the table, either packets or users. This option will be removed in later versions of SVSDM.

Furthermore the object `notifboard` provides the following functions:

- `notifboard.start():`
  starts the notification board.
- `notifboard.rebuildTable(restrictions):`
  rebuilds table with a new restriction object.
- `notifboard.finish():`
  finishes the notification board.

II.4.3.4. Class Life Beat Checker

The object `LifeBeatChecker` is created using following call:

```
lbc = new LifeBeatChecker(connection,
                          status,
                          properties,
                          notifboard)
```

Example 6: Call to create new Life Beat Checker

The submitted parameters represent the following:

- `connection:`
  encapsulates the link to the underlying CORSO software.

- `status:`
  element on GUI giving feedback to the user, will represent the status in two ways: a colour box (green – online, orange – testing, red – offline) and a text representation.
- `properties:`
  represents the configuration file and is needed to retrieve the SVSDM specific information.
- `notifboard:`
  represents the notification board and is needed to initiate the rebuilding of the table, when the connection is available again.

Furthermore, the object `lbc` provides the following functions:

- `lbc.start():`
  starts the life beat checker.
- `lbc.finish():`
  finishes the life beat checker.

II.4.3.5. Class Thread Garbage Collector

The object `ThreadGarbageCollector` is created using following call:

```
tgc = new ThreadGarbageCollector(connection,
                                 properties,
                                 notifboard)
```

Example 7: Call to create new Life Beat Checker

The submitted parameters represent the following:

- `connection:`
  encapsulates the link to the underlying CORSO software.
- `properties:`
  represents the configuration file and is needed to retrieve the SVSDM specific information.
- `notifboard:`
  represents the notification board and is needed to retrieve the content of the table and to initiate the deletion of the timed out packets.

Furthermore, the object `tgc` provides the following functions:

- `tgc.start():`
  starts the garbage collector.
- `tgc.finish():`
  finishes the garbage collector.

II.4.3.6. Interface Packet

The interface `Packet` is used to create new packets for the use in the GPTS or the LPTS, this may happen like this:

```
packet = new Packet(sender,
                    receiver,
                    description,
                    data)
```

Example 8: Call to create new Packet

The submitted parameters represent the following:

- `sender:`
  holds the user ID of the sender.
- `receiver:`
  specifies the user ID of the receiver.
- `description:`
  placeholder to transmit other information, e.g. name of the packet
- `data:`
  will hold the data in bytes, can, for example, be a file.

Furthermore, the object `packet` has to implement the following functions:

- `packet.getBytes():byte[],`
  `packet.setBytes(byte[]):`
  read or write bytes (e.g. from/to a file) to or from the packet.
- `packet.getDescription():String,`
  `packet.setDescription(String):`
  get or set description from/to the packet.

- `packet.getFileInformation():String,`
  `packet.setFileInformation(String):`
  read or write file information from/to the packet.

- `packet.getProfile():String,`
  `packet.setProfile(String):`
  get or set the profile, that represents the receiver's ID in the implementation as available now.

- `packet.getSender():String,`
  `packet.setSender(String):`
  read or write sender user ID.

- `packet.getTitle():String,`
  `packet.setTitle(String):`
  get or set title from/for the packet.

II.4.3.7. Interface Restrictions

The interface `Restrictions` is used to create restrictions for the notification board, this may happen like this:

```
restrict = new Restrictions(sender,
                            profile,
                            description,
                            status)
```

Example 9: Call to create new Restriction

The submitted parameters represent the following:

- `sender:`
  specifies a restriction for the sender ID.

- `profile:`
  sets a limitation for the receiver ID.

- `description:`
  defines restrictions for the description.

- `status:`
  restricts view to the packets of the status mentioned.

Furthermore, the object `restrict` has to implement the following calls:

- `isNotificationNeeded(oid, sender, profile, description, status):Boolean:`
  returns true for the packet if it is put into the notification list.
- `isAuthorizedToBeShown(oid, sender, profile, description, status):Boolean:`
  controls whether the packet will be shown or not.
- `isNotificationNeededTillEnd(oid, sender, profile, description, status)`
  returns `true` if a notification is needed until the end of the lifetime of this object.

II.5. A classical solution

The transportation system described above could also be achieved with the help of a different technology. A database (DB) would play an important role.

In this case the DB would represent the space and keep the data of all objects. This database would have to be installed on a single server. This is important, because a distributed DB outstretched over all workers would lead to partial loss of data as long as some of the workers were not online all the time.

The so called GPTS would be made available on this central server. This process needed to be able to insert, delete, and update table entries by using the SQL[18] syntax. The logging and monitoring feature would be provided by the DB manager.

The difficult part in this setup would be the design of the worker's process that could be achieved by using a web browser. The server could be established by a http-server, where all the clients might log onto in a manner secured by https with username and password. Subsequently the server should create a list of the available packages according to the worker's profile. Then the worker might mark the packages she/he wants to work on for downloading. Afterwards the downloading process would have to

[18] **SQL** (**Structured English QUEry Language** originally SEQUEL, later abbreviated to SQL) is the most popular computer language used to create, modify and retrieve data from relational database management systems. [Rech99]

start, perhaps using a download manager to mask network failures. The files would be saved into a special folder, the additional parameters like time-to-live and execution options might be filed too. Having completed the download, the worker would have to post a request that the copied work packages at the DB would be marked as selected. After being answered with a positive acknowledgment sign, the worker might go offline to work on the packages. The answer packages might be uploaded the same way, in sequence of authentication using https with a username and password, followed by the upload of the files, and waiting for the acknowledgment. Finally the packages might be marked as finished.

The main question in distributed computing is whether:

- wanting to develop a distribution algorithm oneself, which will take quite a long time, as having to deal with a lot of difficulties (some of them are mentioned in the description of SVSDM - see Chapter II.4.2 Inside view)
- or using a middleware that was developed specially for this task and supplies many features already implemented.

A classic solution as described above might use a larger amount of data transfers resulting in more online time/transfer volume and finally higher operating costs.

III. Use Case for an Insurance Company

III.1. Overview

This real life use case was developed together with a German Insurance Company (AMB). They were looking for a new way of overlooking their mobile agents. Until today this task is solved using a special kind of mail system.

The company has existing predefined work tasks that should be dealt with by the mobile workers. These tasks may be defined as workflows that then will be instantiated and populated with specific data. Afterwards, these single packages are ready to be distributed to a mobile worker who may work on these cases.

Examples for such a work task would be: Checking a car after damage, or proving the correct description of a damage report after floodings, or visiting a client to work on a new insurance application etc. These workflows can be predefined on the system, later be populated with the client specific data and then be posted to a work list.

This list can be read by the mobile workers who then take a suitable work package. This step of selecting a work package as talking about a large Insurance Company has to be supported by an automatic process. Therefore each of the predefined work packages has to display a description of skills that are required to fulfill the job. This is supposed to happen in a "Semantic Web" style (see Chapter III.3.5 Profiles), which means some kind of a semantic service description. In addition each worker will show a self defined profile where she/he can tell her/his preferences and a company defined profile which will somehow rate her/his competence for each specific work task or group of tasks.

This automatic process should make sure that these work tasks are always processed by the most capable worker available without making this decision process too complicated.

Such a system should be used by a group of up to 20.000 mobile workers who are organised in agencies. Each agency will have a supervisor, who wants to overlook the work progress of her/his workers in a comfortable way. The agencies also have to report back to the main Insurance Company. This reporting function should also work automatically without any bias.

This is a vision for a complete system serving the insurance company. The functions are quite complex as combining of distribution with central control and semantic assignment of the work tasks is required. For the development of a prototype this thesis concentrates on the distribution mechanism, sharing data packets between users.

III.2. Situation

III.2.1. Actual State

This chapter will describe how the process of acquisition is solved today. It will show the problem of the decentralized collection of work items.

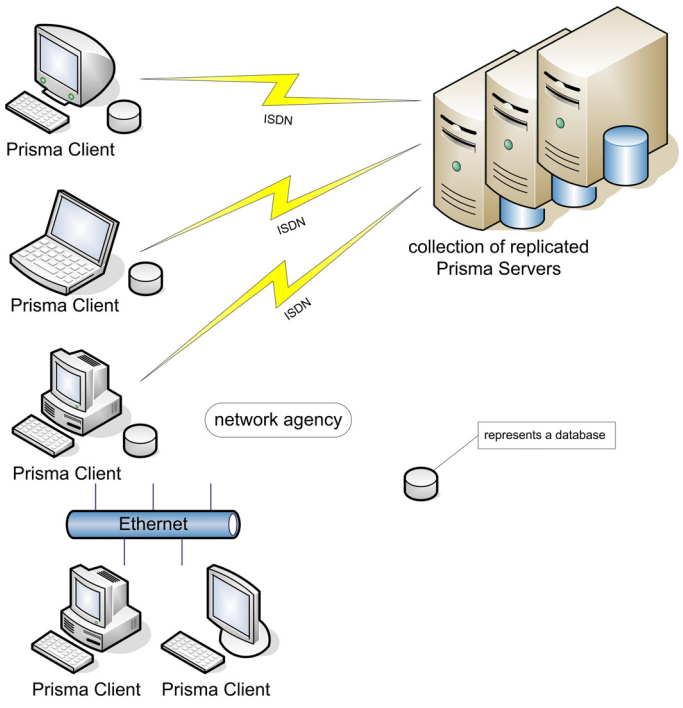

Figure 5: Actual State

The decentralized mobile workers who are using a prisma client (see Figure 5 and Figure 6) collect their work in local databases, or - if more than one mobile client is joined by a network - they may share one database. A client may have more than one user. After the collected work items are approved for sending, they will be put as text files (in the GDV[19]-format) into the outbox. All data for a specific user will be transferred to the corresponding inbox on the server. The connection must be built and the transfer process has to be started manually. The prisma server (for more information on this topic see [www05]) has an in- and outbox for each user.

The data is transferred via FTP[20] to the host and then it is processed. The data connections are represented as arrows in Figure 6.

At the same time new data is sent to the client. This data transfer can be used for all kinds of data, i.e. software upgrades or patches. After the transfer of data is completed client and server will start some automatic or semi-automatic (user interaction being necessary) task that will process the data locally.

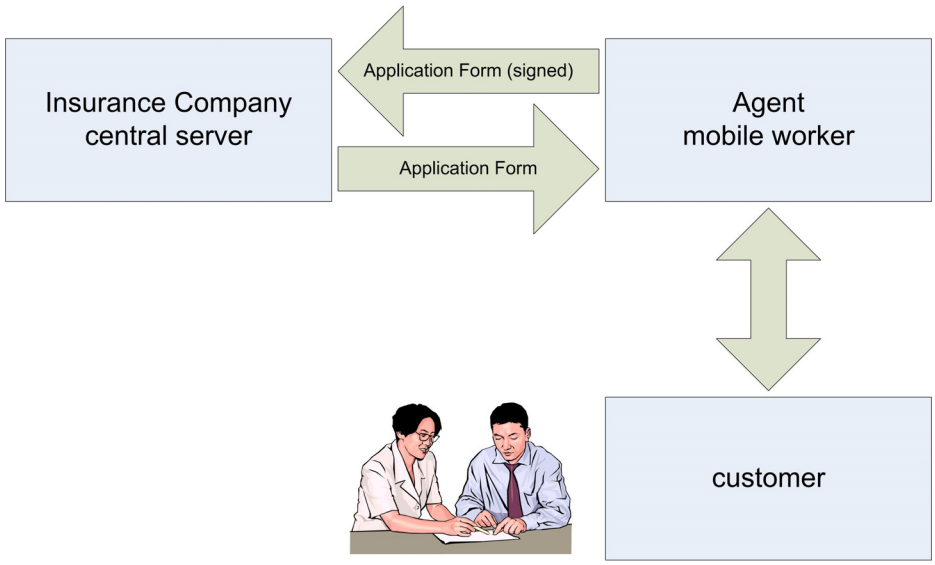

Figure 6: Data-Flow for the Prisma System [www05]

[19] **Gesamtverband der Deutschen Versicherungswirtschaft (GDV)** German for Association of German Insurance Companies. They have set a standard format for the exchange of data between insurance companies or agencies. [www04]

[20] The **File Transfer Protocol (FTP)** is a software standard for transferring computer files between machines with widely different operating systems. It belongs to the application layer of the Internet protocol suite. [Rech99]

Each client (and user) transmits its work tasks independently to the central server. The superior agents or chiefs will be informed about the work of the agents after new contracts are finished. They have no possibility of monitoring the tasks their dependent agents are currently working on.

This hierarchy of agents and superiors can include up to five levels and is not necessarily identifiable via the network topology.

III.2.2. Problems

That description gives a lot of aspects to think about. The following problems arise from the current way of data transmission:

- First the superior agents or chiefs are not quite happy with the current state described above. They can only find out about the work progress of their agents after the work has been done and are not informed of who received which task at what time. The superiors would rather like to have something like an automatic statistic, where they can easily find out about the reliability and sedulity of their agents.
- Another problem to be solved is that the supervisors in the description above do not have any influence on the distribution of the work packages. The challenge is to find a way of how to assign the work tasks to users by defining the skills needed for the work package and allot them according to the workers' abilities. I will discuss solutions for that in Chapter III.3.5 Profiles.
- Another problem is known under the term "synchronizing to hell". This means that most of the data is synchronized and copies are made between the clients no matter, whether they are really needed. Therefore it would make sense if only the necessary data is copied between the clients.

III.2.3. Supported Fields and Persons

The following persons or fields should be supported by the application:

- Information Provider is the database of the Central Insurance Company

  Work Packets may be provided from:

  - Central Insurance Company
  - supervisor of an agency
  - agent herself/himself

- Information Consumer is the agent
- Agency supervisor is controlling and monitoring the workflow
- Final target of all processed packages is the database of the Central Insurance Company

III.2.4. Scenario

For the IT solution of the data transmission it is of great importance that the organisational structures will not be changed in general. The supervisors, however, should get more information about what their agents do when. Most important is that this information flow is achieved without inhibiting the data flow through obstacles or creating a potential bottleneck. There are two ways to reach that aim:

The first way was tried by the insurance company. They simply changed the system so that all agents had to report first to their supervisors. Afterwards the supervisor in charge was obliged to forward the work tasks back to the central server.

This way of solving the previously mentioned deficiencies is not preferable as the supervisor becomes the bottleneck creating new delays in the work process.

A better way to satisfy the supervisors' need of control is to inform them about all the actions of the agents. This may again be done in two ways: either a message is posted automatically to the supervisor or she/he has to look actively into a page of statistics.

In these scenarios all participants in the hierarchy, which may have up to five steps, will be informed.

Both versions could not easily be tested prior the new development as the software currently in use by the Insurance Company does not support automatic messaging or statistic summaries.

Figure 7 shows the second solution. The central server still holds the database, as the Insurance Company is very interested in having all data collected at one point. The agents are still autonomous, but they now automatically inform the supervisor about their actions, which is indicated by the connecting lines. A director has been introduced to show that different hierarchies exist. She/He may be informed by the supervisors.

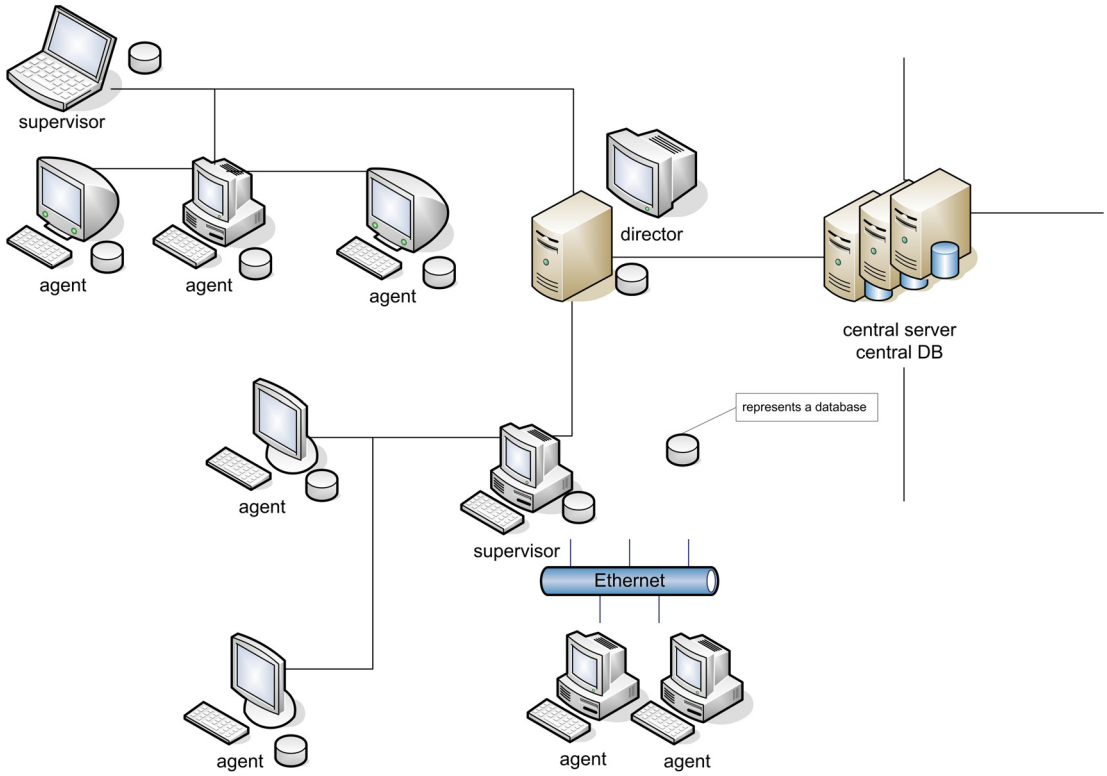

Figure 7: Scenario

III.3. Prototype

III.3.1. Requirements

The following requirements were defined in cooperation with the relevant employee of the IT department at the insurance company and should help to fulfill the task described.

The prototype was restricted to the distribution process only in order to show the benefits of my solution already in this part of the procedure. For this reason the prototype will only transfer data packets between agents, or between agent and server.

Therefore the expenditure of programming work stays limited until the company will decide to follow the path further.

In the following chapters the requirements for the applications are given. Three main programs may be distinguished. The *producer* part generates the packets out of the local database that will be later stored in the central database. The *consumer* receives the packets destined for it from the central database and imports them back into the local DB. Finally the *central database* should work as a well known, always available data source, where the producer saves packets and the consumer may collect them. This last application will also implement the monitoring and statistical functions.

III.3.1.1. Producer

The producer is the part of the system which creates the data packets destined for one specific agent. The following enumeration lists the actions as the producer program provides them for its users.

1. Export of requested data (i.e. acquisition- or inventory data) from the local database into a text file saved to a folder monitored by the producer. This functionality is provided by the insurance company, and so they adapt the export functionality of their database.
2. "Pack" (see Figure 4 above) the package into the LPTS. The producer program should do this automatically, by monitoring the directory where the export process has saved the files.

3. As soon as packets are available to be transferred from the LPTS of the producer to the GPTS at the central database, the connection to the central server is established automatically.
4. Transfer of the data from the LPTS to the GPTS.
5. Disconnection of the producer from the central server.

III.3.1.2. Consumer

The user logged in at the agency acts as consumer of the previously produced packages. She/He imports them into the local database of the insurance agency. The following enumeration lists the actions that the consumer program provides for users.

1. A connection to the central server is set up.
2. The server SVSDM application recognizes the client and transfers new data for the user into the LPTS of the consumer application.
3. The received data is automatically or manually imported into the local database. Therefore the insurance company provides a special import GUI.
4. Disconnection from the central server. This could happen automatically after the end of the transmission (or after step 2, if the connection is not used any longer for other purposes).

The consumer part may be executed on all clients for which the packet may be addressed. (The Prototype will only be able to address one receiver. See Chapter III.3.5 Profiles for an additional addressing system.)

III.3.1.3. Additional Information

Data which are supposed to be exported at the producer are saved into a password protected ZIP[21] file. The user chooses the password during the export procedure, so it is unknown to the system. This way was chosen, because the export procedure does not

[21] The **ZIP** file format is the most widely-used compressed file format in the IBM PC world. The format was designed by Phil Katz for PKZIP, and in the form now applied (PKZIP 2 format) it employs his DEFLATE algorithm for compression. [www01]

support a different security scheme. Later it may be possible to think about a better way of securing the data for example using Pretty Good Privacy (PGP)[22].

The name of a compressed file is, for example, `Q183311.zip`. It specifies the contents of the packet. `Q` (aQuisedaten – German for acquisition data) stands for acquisition and the number code `183311` for the user. `G` (Gesamtbestand – German for inventory data) would stay for inventory data.

The receiver is not a machine but a person who is going to work on this task.

Each computer has to register its users. The file does not reveal the sender. To solve this problem the producer application can read the user code in the `ABAKUS.INI` file (see Example 10) of the Windows[23] system directory. This file represents the configuration for the local database program. It has an entry called `LAST_MRKML` in the section `[Amkas]`, which has as value the user ID. The central server functions as a well known global storage.

As soon as data arrive at the target system, the import into the local database has to be started. For that the application `c:\adsystem\amkas\ImpExpGui.exe` with three space separated parameters has to be started.

1. the user ID (the numeric part of the filename, i.e. `183311`)
2. a fixed `1` (the meaning of this parameter is not known)
3. the KU (KonzernUnternehmen – German for allied company) sign. This is a numeric value between one and six. It can be found in the `ABAKUS.INI` section `[Amkas]` at the entry `KU`.

A valid call would be:

```
C:\adsystem\amkas\ImpExpGui.exe 183311 1 5
```

---

[22] **PGP** is a computer program which provides cryptographic privacy and authentication. PGP was originally designed and developed by Phil Zimmermann in 1991. [Dust03]

[23] **Microsoft Windows** is a range of commercial operating environments (e.g. Microsoft Windows NT or Microsoft Windows XP) for personal computers. The range was first introduced by Microsoft in 1985 and eventually has come to dominate the world personal computer market. All recent versions of Windows are fully-fledged operating systems. [www01]

The import application is not able for batch processing yet (This feature will probably be available at the end of 2005). The above call will open a GUI, where the user will have to fill-in the password and start the import procedure.

The KU-sign may be read either from the sending or receiving host, but if the producing client reads the user ID of the sender from the configuration file, it would be one step to get the KU-sign from there as well.

```
[Amkas]
...
LAST_MRKML=183311
...
KU=5
...
```

Example 10: Example for abakus.ini

III.3.2. Solution

Finally the following solution (see Figure 8) is agreed upon as being the best way of testing the infrastructure with this new kind of technology. The prototype can be integrated into the old database system to improve the way of data exchange even for mobile devices, which are offline most of the time. Furthermore it adds the possibility of monitoring and statistic analysis, because of the fact that all actions are logged centrally.

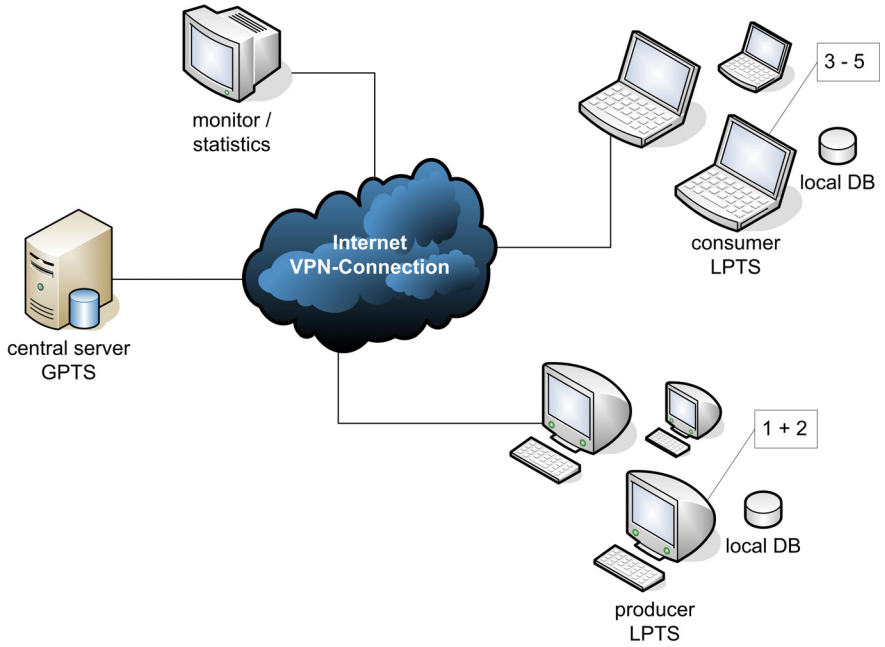

Figure 8: Proposed solution for the insurance company

Steps of the solution process as depicted in Figure 8:

1. Export of data into the directory monitor by the producer. The data is saved as a file, its name indicates what kind of data it is (`Q` stands for acquisition, `G` stands for inventory). The numeric part explains, who is supposed to be the receiver of the packet.

   Responsible: insurance company/user.

2. Producer application listens to that directory. If new files (i.e. `Q183311.zip`) are added these will automatically (in case the connection to the central server is available) be uploaded to the GPTS. If no connection is available the packet will be stored in the LPTS of the producer and as soon as possible then uploaded to the GPTS.

   Responsible: SVSDM application.

3. Consumer sets up a connection to the central server (to the GPTS). Then the Consumer application is notified about new packages available.

   Responsible: insurance company, SVSDM application.

4. Automatic download of all packets for user `183311` into a predefined local directory.

   Responsible: SVSDM application.

5. Start of the Import GUI (`ImpExpGui.exe`) with the correct parameters. The user finishes actively the import process.

   Responsible: SVSDM application, insurance company.

III.3.3. User Interface and Operation

All three user interfaces were implemented with special attention to the user's needs. They all have some kind of a status indication that is put into practice like a traffic light which reports to the user the current availability of the online connection. *Red* means offline, *orange* means testing for the connection (unsure), and *green* stands for online. This kind of representation was designed to provide an easy way of understanding even for a user unaware of the technical background.

For each program a configuration file (see Example 11, Example 14, Example 16) is available. The following parameters are important for the functionality of the shared virtual space and can be found in most of the three configuration files:

- `svs_sitename`:
  defines the site name, where CORSO runs showing a symbolic name or an IP-address. The local site name can be retrieved and used for the address, too.
- `svs_siteport`:
  the internet-stream port number of CORSO at the above defined site (for example 5005 for the standard port of CORSO).
- `svs_domain`:
  This parameter is only used on Windows platforms. It specifies the domain of the system.
- `svs_user`:
  the user-ID used to login to CORSO.
- `svs_password`:
  the password corresponding to the user-ID.
- `svs_id_LPTS`:
  name for the personal LPTS of the worker. This allows the set-up of different LPTS's for different workers on one single computer by simply changing this ID.

- `svs_id_GPTS`:
  name of the GPTS, must be the same on all applications that cooperate with the same global storage.
- `svs_com_GPTS`:
  name for the communication object which is used to coordinate the packet's transmission to the GPTS. This must again be the same on all applications that cooperate on the same global storage.
- `svs_ip_GPTS`:
  stands for the IP-Address or site name of the host where the GPTS is running (for example 192.168.0.1, that must be a valid IP-Address, or site name).

III.3.3.1. Producer

The Producer (Figure 9) uploads the packets via the SVSDM-middleware. The producer program listens to a directory defined in the configuration file named `producer.ini` (see Example 11).

It may be switched between two operational modi:

- One is semi-automatic (option `auto_transfer` is set to `no`, see Example 11). It enables the view of what is happening on a graphical interface (GUI) and as soon as wanting to start the upload process the `Transfer` Button has to be pressed.
- The second mode is fully automatic (option `auto_transfer` is set to `yes`). If selected, all files found in the directory will be pushed up into the GPTS. In this second mode the GUI may be disabled as well (option `gui` is set to `no`), which renders the distribution to the GPTS fully unobservable.

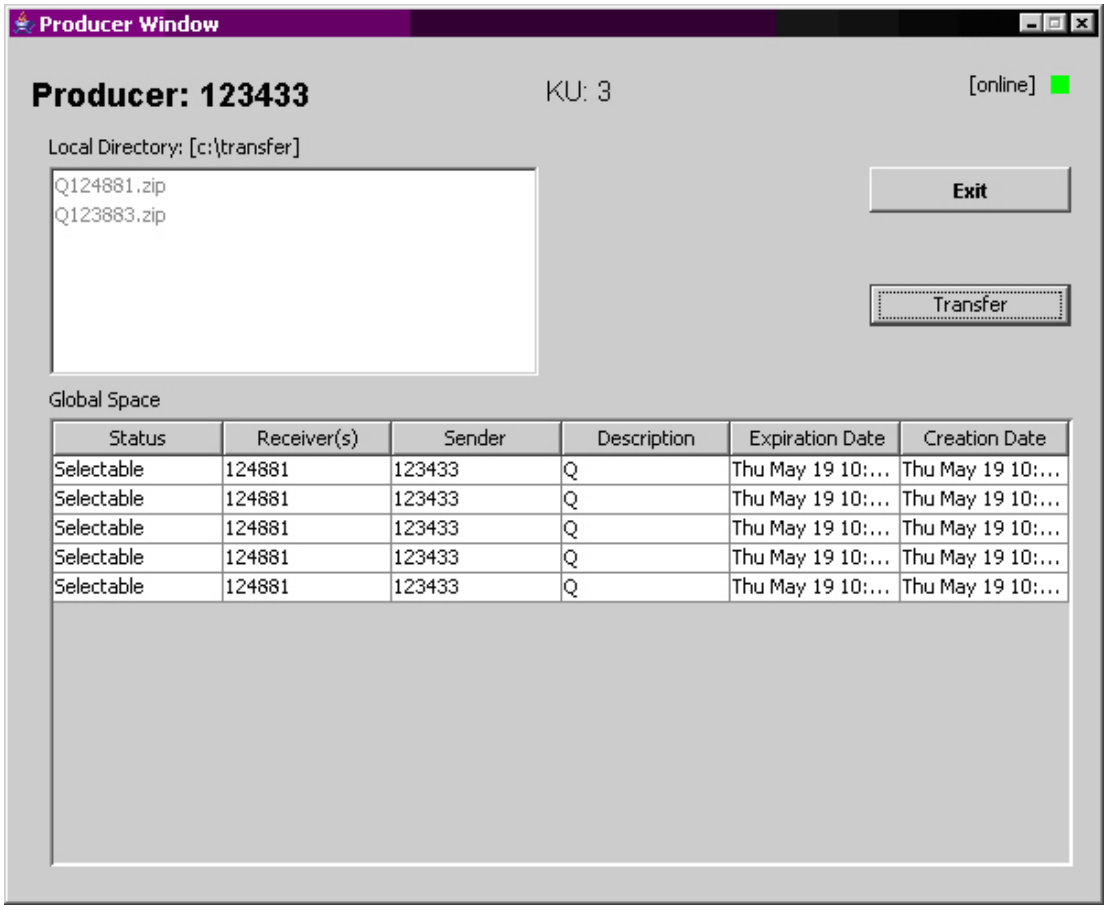

Figure 9: Producer Window

The configuration file (Example 11: Example for producer.ini) sets various parameters. See the following list for the special parameters of the `producer.ini` and their meaning:

- `id`:
  sets the user ID, but is ignored if the `abakus.ini` is found, and has a valid `LAST_MRKL` entry - for more information see Chapter III.3.1.3.
- `ku`:
  sets the `ku` sign; - the same restrictions apply as for the `id`.
- `directory`:
  location of the directory where the files of the local database are saved.
- `abakus.ini`:
  path to the file which will hold the `id` and `ku` sign.
- `gui`:
  controls whether the GUI is shown or not. If `no` is selected `auto_transfer` is set to `yes` ignoring the actual setting.

- `auto_transfer`:
  if set to `yes` new packets are uploaded to the LPTS automatically and are forwarded to the GPTS as soon as an online connection is available.

```
#producer Property File generated automatically
#Wed May 04 13:56:58 CEST 2005


id=123433
ku=3
directory=c\:\\transfer
abakus.ini=C\:\\WINDOWS\\abakus.ini

gui=yes

auto_transfer=no

svs_sitename=localhost
svs_siteport=nnnn
svs_domain=none
svs_user=Corso User
svs_password=none
svs_id_LPTS=lpts_packet
svs_id_GPTS=root_packet
svs_com_GPTS=root_comm
svs_ip_GPTS=xxx.xxx.xxx.xxx / hostname
```

Example 11: Example for producer.ini

Example 12 shows the pseudo code for the `producer`: First the connection to the underlying CORSO software is established (see line 3). Then the LPTS is created (lines 5-14). For the meaning of its creation parameters see Chapter II.4.3. Then the restrictions for the notification board are created (line 16 and 17). They filter that only packets from this user shall be visible. Afterwards the two notification boards are created (lines 19 to 37), where only the one of the GPTS is visible on the GUI, because the packets stay only for a short while in the LPTS as they are uploaded as soon as possible. Finally the `LifeBeatChecker` (line 39 and 40) which visualizes the online status and the `FolderThread` (line 42 and 43) which checks the import directory are started.

```
producer(properties, list, loc_table, glo_table, status) {
    //get CORSO Connection
    connection = getCorsoConnection(properties);
    //create LPTS
    try {
      lpts = new LocalPTS(connection, getProperty("svs_id_LPTS"),
                          getProperty("svs_id_GPTS"),
                          getProperty("svs_com_GPTS"),
                          getProperty("svs_ip_GPTS"), true,
                          getProperty("id"));
    }
    catch (Exception) {
      error("could not create LPTS.");
    }
    //create restrictions
    loc_restr = new Restrictions(getProperty("id"), "", "", ALL);
    glo_restr = new Restrictions(getProperty("id"), "", "", ALL);
    //create NotificationBoard for LPTS, will not be shown on GUI
    try {
      local = new NotificationBoard(connection, loc_restr, loc_table,
                                    getProperty("svs_id_LPTS"),
                                    getProperty("svs_sitename"), true);
      local.start();
    }
    catch (Exception) {
      error("could not create local NotificationBoard");
    }
    //create NotificationBoard for GPTS
    try {
      global = new NotificationBoard(connection, glo_restr, glo_table,
                                     getProperty("svs_id_GPTS"),
                                     getProperty("svs_ip_GPTS"), true);
      global.start();
    }
    catch (Exception) {
      error("could not create global NotificationBoard.");,
    }
    //create LifeBeatChecker
    lbc = new LifeBeatChecker(connection, status, properties, local);
    lbc.start();
    //create FolderThread
    ft = new FolderThread(list, properties, lpts);
    ft.start();
} //end
```

Example 12: Producer Code

```
1  FolderThread(list, properties, lpts) {
2      //check which mode should be active
3      auto_transfer = getAutoTransferStatus();
4      //check directory forever
5      do {
6        directory = getProperty("directory");
7        list.setListData(directory);
8        //if transfer is true try to transfer file
9        if (transfer) {
10         for (all files in directory) {
11           if (file.canRead() && file.canWrite() &&
12               file.getType.equals("zip")) {
13             echo(file+" is read/writeable, will be pushed into space.");
14             //create packet and insert it into LPTS
15             try {
16               packet = new Packet(sender, receiver, description, file);
17               packOID = lpts.placeInLPTS(packet, ttl);
18             }
19             catch (Exception) {
20               if (packOID != null) {
21                 try {
22                   lpts.deletePacket(packOID);
23                 }
24                 catch (Exception) {
25                   error("error while exception handling at LPTS insert.");
26                 }
27               }
28               showErrorMsg("error at LPTS insert.");
29               continue;
30             }
31             //after successful insertion delete file
32             file.delete();
33           }
34         } // for
35       } // if (transfer)
36       transfer = auto_transfer;
37     }
38     while (forever);
39 } //end
```

Example 13: FolderThread Code

Example 13 shows in pseudo code the function of the `FolderThread`: This thread is started by the producer application in the background and its task is to listen to a directory (line 7 fills a list element of the GUI with the contents of the directory to give the user feedback) and then insert all of the ZIP-files found there that are currently readable and writeable (see lines 11 and 12 for this check) into the LPTS (see line 17). This may happen in two different modi (see above for more information). Before this insertion the packet supposed to hold this file (see line 16) has to be created. After successful insertion of the packet into the LPTS the original file is deleted (see line 22). If an exception happens the file is kept and is tried once more (see lines 19-29).

III.3.3.2. Consumer

The Consumer (Figure 10) receives packets that carry the ID of the user logged on to the user interface. If an online connection is established all dedicated packets are automatically downloaded to the LPTS. It may be selected either to see `All`, only `Finished` or `Selected` packets on the table of the GUI.

All packets that have not been transferred to the external import program are marked as `selected`; afterwards they are marked as `finished`. The finished ones are stored until the end of the time-to-live (option `TTL_LPTS` is specified in milliseconds, see Example 14).

It is again possible to switch between two modi:

- One is semi-automatic (option `auto_transfer` is set to `no`). The starting of the import program has to be done manually (button `Start selected`)
- Automatic (option `auto_transfer` is set to `yes`). After that all new packets are sent automatically to the import program. In this mode the GUI may be disabled completely (option `gui` is set to `no`).

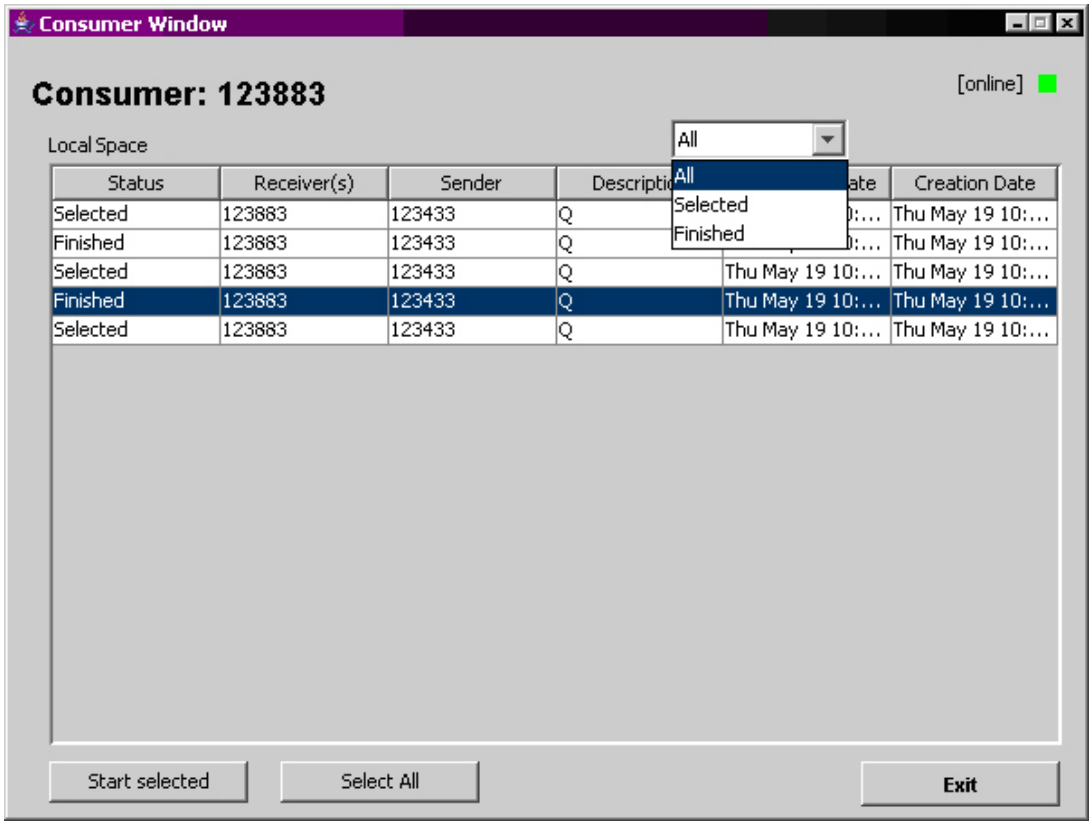

Figure 10: Consumer Window

The configuration file (Example 14: Example for consumer.ini) sets various parameters. The following are specific to the consumer:

- `id`:
  sets the user ID.
- `directory`:
  location of a temporary directory where the files for the local database will be saved.
- `external_program`:
  gives the system path to the external import program which will be called to handle the contents of the packets.
- `TTL_LPTS`:
  time-to-live of the packets in the LPTS (storage timeout).
- `gui`:
  controls whether the GUI is shown or not, if `no` is selected `auto_transfer` is set to `yes` ignoring the actual setting.
- `auto_transfer`:
  if set to `yes` arriving packets will be automatically handed onto the import program.

```
#consumer Property File generated automatically
#Fri May 06 10:45:16 CEST 2005

id=123883
directory=c\:\\import
external_program=notepad
TTL_LPTS=1000000

gui=yes
auto_transfer=no


svs_sitename=localhost
svs_siteport=nnnn
svs_domain=none
svs_user=Corso User
svs_password=none
svs_id_LPTS=lpts2_packet
svs_id_GPTS=root_packet
svs_com_GPTS=root_comm
svs_ip_GPTS=xxx.xxx.xxx.xxx / hostname
```

Example 14: Example for consumer.ini

```
1  consumer(properties, loc_table, glo_table, status) {
2      //get CORSO Connection
3      connection = getCorsoConnection(properties);
4      //create LPTS
5      try {
6        lpts = new LocalPTS(connection, getProperty("svs_id_LPTS"),
7                            getProperty("svs_id_GPTS"),
8                            getProperty("svs_com_GPTS"),
9                            getProperty("svs_ip_GPTS"), true,
10                           getProperty("id"));
11     }
12     catch (Exception) {
13       error("could not create LPTS.");
14     }
15     //create restrictions
16     loc_restr = new Restrictions("", getProperty("id"), "", ALL);
17     glo_restr = new Restrictions("", getProperty("id"), "", ALL);
18     //create NotificationBoard for LPTS
19     try {
20       local = new NotificationBoard(connection, loc_restr, loc_table,
21                                     getProperty("svs_id_LPTS"),
22                                     getProperty("svs_sitename"), true);
23       local.start();
24     }
25     catch (Exception) {
26       error("could not create local NotificationBoard.");
27     }
28     //create NotificationBoard for GPTS, will not be shown on GUI
29     try {
30       global = new NotificationBoard(connection, glo_restr, glo_table,
31                                      getProperty("svs_id_GPTS"),
32                                      getProperty("svs_ip_GPTS"), true);
33       global.start();
34     }
35     catch (Exception) {
36       error("could not create global NotificationBoard");
37     }
38     //create LifeBeatChecker
39     lbc = new LifeBeatChecker(connection, status, properties, global);
40     lbc.start();
41     //create ThreadGarbageCollector
42     tgc= new ThreadGarbageCollector(connection,properties,lpts);
43     tgc.start();
44 } //end
```

Example 15: Consumer Code

Example 15 shows in pseudo code the function for the `consumer`: First the connection to the underlying CORSO software is established (see line 3). Then the LPTS is created (lines 5-14) for the meanings of the creation parameters see Chapter II.4.3. Thereafter the restrictions for the notification board are created (line 16 and 17). They filter the packets in a way which makes visible only those from the same user. Afterwards the two notification boards are created (lines 19 to 37), but only the one depicting the LPTS will be visible on the GUI, because the packets will only stay for a short while in the GPTS as they are supposed to be downloaded to the consumer's LPTS as soon as possible. Finally the `LifeBeatChecker` (line 39 and 40) which visualizes the online status and the `ThreadGarbageCollector` (line 42 and 43) which watches the packets and deletes the timed out packets are started.

III.3.3.3. Monitor

The Monitor (Figure 11) is involved according to the needs specified in the requirements. The Central Insurance Company wanted to have the possibility of monitoring all the events in the system. For better practicability a GUI is implemented. In the environment of the prototype the monitor also creates the GPTS (button `Start/Stop Services`). It shows all registered users and allows one to add or delete user IDs. All packets that are stored in the GPTS may be seen and one may add or delete packets there as well. The last item was actually a development feature, but it was kept in the system for administrator usage.

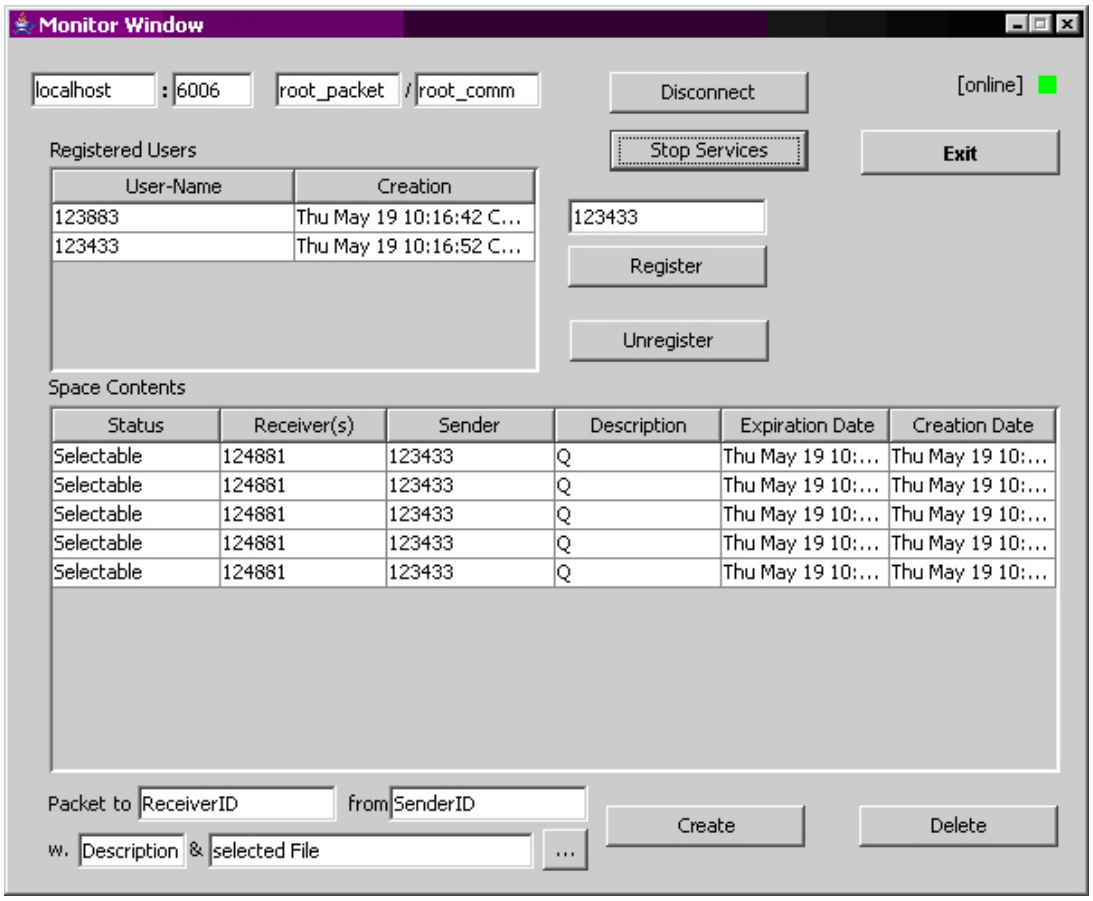

Figure 11: Monitor Window

```
#monitor Property File generated automatically
#Wed May 11 22:31:05 CEST 2005

TTL_GPTS=1000000


svs_sitename=localhost
svs_siteport=nnnn
svs_domain=none
svs_user=Corso User
svs_password=none
svs_id_GPTS=root_packet
svs_com_GPTS=root_comm
```

Example 16: Example for monitor.ini

The configuration file (Example 16: Example for monitor.ini) sets various parameters. The following attribute is specific to the monitor:

- `TTL_GPTS`:
  time-to-live of the packets in the GPTS (storage timeout).

```
1  monitor(properties, space_tb, user_tb) {
2    //if connect is pressed
3    public void connect() {
4      //get CORSO Connection
5      connection = getCorsoConnection(properties);
6      try {
7        //create GPTS
8        gpts = new GlobalPTS(connection, getProperty("svs_id_GPTS"),
9                             getProperty("svs_com_GPTS"), properties, true);
10       //create NotificationBoard for GPTS
11       space = new NotificationBoard(connection,
12                                     new Restrictions(), space_tb,
13                                     getProperty("svs_id_GPTS"),
14                                     getProperty("svs_sitename"), true);
15       space.start();
16       //create NotificationBoard for user list
17       users = new NotificationBoard(connection,
18                                     new Restrictions(), users_tb,
19                                     getProperty("svs_com_GPTS"),
20                                     getProperty("svs_sitename"), false);
21       users.start();
22     }
23     catch (Exception) {
24       error("could not create GlobalPTS or NotificationBoards.");
25     }
26   }// connect()
27   //if start Services is pressed
28   public void startServices() {
29     try {
30       //start GPTS services
31       gpts.startServices();
32       //create LifeBeatChecker
33       lbc = new LifeBeatChecker(connection, status, properties);
34       lbc.start();
35     }
36     catch (Exception) {
37       error("starting of services failed.");
38     }
39   } //startService()
40 } //end
```

Example 17: Monitor Code

Example 17 shows in pseudo code the function for the `monitor`: This code is separated into multiple functions, as the GUI gives the user the possibility of administering different global storages (it is possible to give the host, id_GPTS – name for the GPTS, and com_GPTS – name of the communication object – on the user interface). Therefore there are `connect()` and `startServices()` which both have a corresponding button on the GUI.

The first function (lines 3 to 26) creates the connection to the underlying CORSO software (see line 5). Then the GPTS is created (line 8 and 9). For the meaning of the creation parameters see Chapter II.4.3. Then two notification boards are created one of which (lines 11-15) represents the global storage, while the second (line 17-21) shows the users currently admitted to use the system.

The second function starts the services provided by the GPTS, that is the GPTS itself (line 31) and again the `LifeBeatChecker` (line 33 and 34) which visualizes the online status.

III.3.4. Demonstration

The mobile worker, in this use case the insurance agent (as consumer), transfers her/his data package onto her/his wearable computer or wireless display unit. This way she/he has access to the data in all locations, even at a customer's home. Afterwards she/he finishes her/his task and return the packet back (being a producer) to the global space (GPTS) for further processing by a different agent or the central office of the insurance company. As this prototype is limited to the distribution of data objects only it is difficult to understand the necessity of such mobile devices but the Use Case for a Mobile Medical Service features a different scenario, where small and handy computing devices are of key importance.

III.3.5. Profiles

In the requirement analysis it was found out that the distribution from one client to another is not necessarily the best option. To solve this problem some additional assignment features (profile management) will be discussed. In the case of the prototype the insurance company only wanted to test the strength of the distribution algorithm, and so complex assignment features are skipped on purpose.

III.3.5.1. Simple profile

The simplest way to achieve an assignment is to name the receiver. This way was implemented in the "Use Case for an Insurance Company" described above. Each packet is destined for one specific agent, she/he will be named with her/his ID in the packet.

III.3.5.2. Hierarchical profile

Company organizational structures are analogous to hierarchical file systems, so the representation of the workers' hierarchy may be set up in a similar manner and be built into the distribution system. This would start like a file system with a root node [/]. Work packets may then be posted to one specific worker, telling the path of how to reach her/him. In the case of sending a package to a logical node representing a higher hierarchy, the packet might be forwarded to all agents available below the given node name. Such a system could be useful for document delivery, where documents may be relevant for a whole organizational substructure.

III.3.5.3. Complex profile (semantic web)

A possible solution to this complex profile was mentioned before. It should be possible to use some kind of skill profile of the workers. One that may be created by the worker, or perhaps parts of it might be validated by the system (the system watches the user's behavior and assigns her/him automatically generated skills) or the head of the agency entitles the worker to certain skills.

Each packet would then carry these predefined attributes that were set up at the packet's creation time. Both property files and user's skills will have to be compared and those users' best fulfilling the demands should be able to download a certain packet. Solutions for similar problems are sought in many semantic web [Stuc05] projects.

IV. Use Case for a Mobile Medical Service

IV.1. Situation

IV.1.1. Actual State

To get a better idea of what this use case is about, a short introduction into the Mobile Medical service of Vienna will be given. This service is available on weekdays from 7:00 pm to 7:00 am and on weekends and holidays the whole day long. It replaces during this time span the general practitioner personally not available for her/his patients.

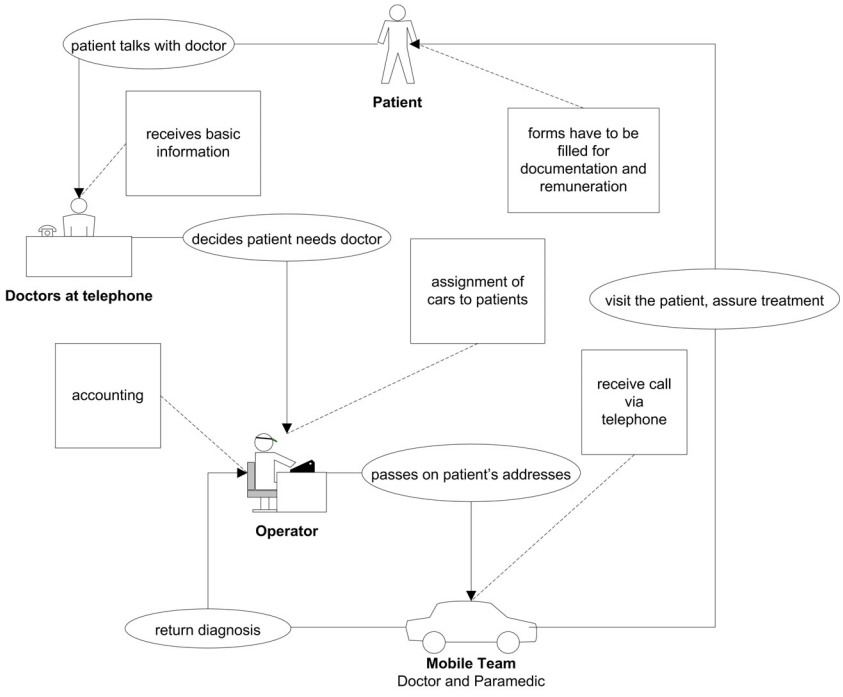

Figure 12: Workflow of Mobile Medical Service

Figure 12 shows how the Viennese mobile medical service works: The patients call a physician at a central office. The doctor in the call centre is the person who may give advice and also estimates the urgency of the case. If the patient needs to be visited by a doctor the name, address, age, and diagnosis of the caller are recorded with the help of a computer program. The physician may decide upon three different degrees of urgency: normal (not a life threatening case, can wait for the doctor up to three hours), urgent (the next free mobile team will take the visit, the team has to be at the address within the next 30 minutes), or "blue" (very urgent, a team has to be at the address as soon as possible. Only teams with emergency cars featuring a blue light may get these cases).

The passing on of the collected data to the mobile teams (mostly a physician with a paramedic) is done by non-medical personnel. They coordinate the mobile teams (in Vienna at night up to three and by daylight (weekends or holidays) up to ten teams) and transmit the data of the patient via voice radio or mobile phone.

Then the mobile team visits the patient. The doctor performs a physical examination of the patient and decides on the next action (which fully lies in the responsibility of the visiting physician). These actions vary between writing prescriptions and sending the patient to a hospital where further examination and treatment can be done. While doing this the physician has to document all her/his actions into a scrapbook for later reference and the paramedic will prepare the forms (e.g. the prescription or the papers for the hospital). Each visit also has to be documented in an assignment book, where the patient's name, address and insurance information are recorded. Then this has to be signed by the patient to confirm the accuracy of the data and by the doctor confirming the medical diagnosis.

Afterwards this diagnosis is submitted back to the central office again using a mobile telephone or voice radio. A non-medical operator adds the received information to the database of the computer program where it is finally saved.

IV.1.2. Problems

The software used in the central office at the moment was developed to help with the data collection process at the call centre and to support the non-medical operator assigning the cases to the mobile teams. The data gathered in this manner is stored for documentation reasons, but it will not be updated or corrected after the visit of the mobile team, when only the medical diagnosis of the attendant physician is added. As

the data exchange is done manually it would take a lot of time to compare the sets of data in the central office with those collected by the mobile teams.

A solution would be to use a system that is capable of letting the mobile team manipulate the actual database so that the changes can be promoted in order to create an error free database.

This takes us to another problematic issue:

The data of the patient is needed to be filled into a number of forms and nowadays this is done by paramedic personnel. In the process of a medical visit the basic patient data need to be copied. This does not only give the possibility of innumerable errors but also wastes valuable time that would better be spent examining the patient or used to finish the visit more quickly. If the mobile system were equipped with a small printing unit, the medical care could be achieved faster and more accuratly.

Another detail to be thought about is the documentation. The mobile physician is obliged to keep her/his own documentation of all cases. Nowadays this is done manually into a scrapbook, but with a fully computerized system this could automatically be provided more comprehensivly in less time.

IV.1.3. Requirements

The scenario described above and its accompanying problems generate quite a lot of requirements. The system could work as follows:

The caller is identified by the computer system and the data known from the telephone company is prefilled into the form before the call is answered. The physician at the call centre then may skip the time-consuming task of collecting these data and would just have to check the data and correct them or submit additional information. If the caller was unidentified or different from the patient this would have to be submitted too. This process of data collection or correction might be skipped in case the patient would not need to be visited by a mobile team, but the call would be logged in any case in order to help doctors see the context should the patient call again. If a medical visit proves inevitable, the doctor records a provisional diagnosis and requests a mobile team. She/He also sets the urgency level to normal, urgent or "blue" (very urgent).

This information is transmitted to the assignment process. This could be supported by an automatic traffic routing software, which may be seen as an addition, not

implemented in the first step; it may still be done manually by the non medical operator. This will never fully be working without human interaction as some decisions are too critical and too difficult to be decided by a computer program. The case may be delegated to the nearest team in charge, but also different factors like the urgency level or the team's workload play a role. The final aim should be to distribute the work evenly (e.g. give all the teams the same number of cases). Finally it is also important to ensure that the teams working for 12 hour periods some breaks for refreshment and lunch.

After the case has been allocated to a team it is digitally transmitted. The mobile team will confirm the assignment of the new case and drive to the given address. Then they have to send a status code telling that they arrived at the patient. This could also happen automatically if the mobile team's system is equipped with GPS[24] supported navigation software. The team might look into, correct, add and work on the patient's data. In this case the basic personal data might also be printed onto forms needed to be handed over to the patient for later reference and further treatment. Therefore a handy mobile printing unit would be very helpful.

The device in use to communicate with the mobile team must comply with the following requirements: it must be mobile, not too heavy, internally powered, robust, always ready for action, handy, cheap, user-friendly, and equipped with an easy-to-use small and weightless printing unit for dealing out the forms. It should produce good printing quality and last for a long time (for such tasks a thermal printer seems inappropriate).

The digital data transfer could be accomplished either by direct connection to the device or by using the functionality of a connected mobile phone.

Furthermore the documentation (a protocol of the visit) could be done more quickly and efficiently, because of the time saved using this device. This documentation will be very much appreciated by the physicians because it will help them to reproduce easily what happened. This documentation may be worked out with the assistance of a computer, so complications or questions can be solved more quickly. By these means

---

[24] The **Global Positioning System (GPS)** is a satellite navigation system used for determining one's precise location and providing a highly accurate time reference almost anywhere on Earth or in Earth orbit. [www01]

parts of the patient file may also be forwarded to the family doctor or the hospital the patient is assigned to.

Another difficulty in setting up such a program is to keep the system open to later extensions.

IV.2. Possible Solution

In the course of analysing the requirements I found out that many sequences of operations are similar to the use case of the insurance company. Yet in this use case the different tasks have always to be executed in the same sequence. Therefore a solution is suggested where a WorkFlow Management System (WFMS) plays the main role. With the help of a WFMS the sequence of actions can be predefined and so the messages with the appended data are sent to the correct client or endpoint. The workflow system supplies the worker with all the data and tools needed for completing her/his job. The WFMS will somehow also have a guiding and scheduling position, as it will make sure that the entered cases go through all predefined steps and are not left unfinished.

IV.2.1. Workflow Management System

A Workflow Management System is a software system that defines, manages, and executes workflows (see Chapter II.2 BPEL). The execution order is defined by the workflow logic.

For this use case a previously created WFMS (see [Riem03]) is used to coordinate the geared order of events more easily.

The WFMS as presented in [Riem03] is proposed because of its simplicity and easy to construct workflows. This leads to fast and not too complex execution. It is limited to *activities* that may have the state of `init` (initialized), `running`, or `done` and *transitions*. The state `init` means that the activity is waiting for the input data. Afterwards the process switches into `running`, which means that the application dedicated to the activity was started or that the dedicated task has been executed and is waiting for an answer. After all partial jobs have been finished the activity might switch to `done`. There is a special activity – the "start-activity" – which will be set to running after the instantiation of the workflow.

Transitions control the sequence of activities. They are the connectors between at least two activities and trigger the switch from `init` to `running`. Two types of transitions can be distinguished:

- *all_of_n*, the transition triggers when all input activities have been completed
- *one_of_n*, means that when the first activity is finished the transition is valid.

Using this WFMS the main workflow (see Figure 12: Workflow of Mobile Medical Service) may be implemented by defining an order of the tasks at the call centre and at the mobile team. The following activities have to be taken care of:

1. start action: new call is coming in at the call centre
2. doctor takes call and new form is popping up with information filled out by the telephone company
3. during the answer of a phone call the physician may chose one of two options:
    a. no mobile doctor is needed, case is ended by medical advice via telephone (go to 1).
    b. physician decides mobile team is needed
4. case is sent to the operators or allocated automatically to a mobile team
5. case is transmitted to the mobile team, then confirmed
6. patient is reached by the mobile team:
    a. examination
    b. diagnosis
    c. treatment
7. control of the data by the patient and the physician, signatures by both of them
8. dealing out filled forms (prescription, letter for hospital, …)
9. finished case, return final diagnosis, get ready for a new case (go to 5)

IV.2.2. Operation

For the overall operation the tools have to be put together. On the one hand there is the WFMS, controlling the sequence of events, and on the other hand there is the SVSDM, the reliable parcel service, which is used to transport data packets. Both parts put together can form a full, functional system:

The telephone call comes in and is routed to the first free place in the call centre. The computer application presents the form for taking a new call to the medical personnel. The telephone number and the data reported by the telephone company are already filled in. The doctor now may take the call and advise the patient of what she/he could do.

If it is necessary to send a mobile team the medical personnel will be guided by the program (actually the specific workflow at the WFMS will do that) through the process of gathering the information needed. If all the information is entered the doctor will set the urgency level and finish the case.

Then the WFMS will forward the case to the next activity, to the operators, who will assign the case to a mobile team. After that the next activity is reached, therefore the data packet of the case details will be sent using SVSDM to the mobile team as soon as they have a connection again.

When the new case is received the mobile team will confirm the reception and will start their visit. After reaching the patient the physician will examine her/him and decide on the next steps together with the patient. If further treatment in the hospital is needed the mobile computing unit will prepare the forms (with the data already known, this may have been corrected before) and print them. Afterwards a report of the visit with the data of the patient has to be signed by the doctor and the patient.

This report will later be forwarded to the Social Insurance and probably a more detailed one will be sent to the hospital and family doctor. Finally the task will be marked as finished and returned by the SVSDM to the central office and the mobile team will be free to take the next assignment.

The SVSDM may play a very important role in transferring the data to and from the mobile units. It is specially developed for use by mobile units. Therefore it keeps the

transfer overhead minimal and contains special features which ensure that the data arrive complete or are discarded.

The WFMS makes sure that the activities or actions are done of in the correct order. It may have a workflow for the whole process but also for some sub-processes that are regularly passed in the same way (e.g. the filling out of the patient's form by the doctor on the telephone).

V. Other Use Cases

V.1. Use Case for a Telecom Service Provider

V.1.1. Situation

The Austrian Telecom employs so-called Service teams which are supposed to repair defects, or to install new products at customers' homes. For this purpose they have to fulfill predefined tasks. Until now these tasks have been taught through cooperation with an experienced colleague, but as technology is evolving fast, the steps are changing quickly. As a result, important steps may be forgotten or omitted.

Therefore a new mobile system could be created; its task would be to support the mobile teams in all situations. It should have detailed descriptions of all usual work tasks. In order to achieve this, a workflow such as a description of all probable tasks should be available for reviewing. The description should be shared amongst all teams and if someone changed a step to simplify the process or because of changing technology, all teams should be informed about these changes.

V.1.2. Possible solution

The solution should be orientated on the use cases described before. An application with an included WFMS like the "Use Case for a Mobile Medical Service" would be really helpful. The workflows would be centrally stored and maintained by specially assigned workers. The most important fact would be that the activities and transitions might be changed by the mobile worker as well. She/He would download her/his work tasks at the beginning of her/his work shift. Therefore a description of the task with a list of all parts needed and the workflow of the installation process would be

transmitted. Before leaving for the customer the parts would have to be acquired. To support this process the mobile unit might present an electronic checklist where the part numbers would be entered. This could automatically trigger, in case of near shortage, the order of new products. Then the mobile team would be ready to leave for the first costumer, and the system could support the driver with navigation information using a GPS system. After arrival the first task would be to compare the data about the costumer, and then the actual installation or repair process could begin. The mobile unit might present a list of steps to be followed to reach the aim of this visit. The checklist might present only the important steps, and if a worker needed more information she/he may get more detailed information on the selected step. At the end of the service a final report could be printed, which would be signed by the worker and the costumer. The data of the material used and time spent could be transmitted back to the company to be used for purposes of the accounting department.

This solution could profit from the SVSDM as a reliable parcel service in favour of a mobile solution. The WFMS would be used as a coordinator to make sure that all activities are carried out, with the addition of a feature to enable the worker to alter the workflow to her/his needs, because of a different service process. These modifications may have different reasons, either because of technology changes or of logical changes to have a better solution.

V.2. Generic Use Case

Three use cases were outlined. The first one was implemented and found acceptance in a company. The other two were described and a possible solution suggested. This experience now leads to the idea to point out which preferences or requirements are necessary to use SVSDM in a certain situation. The similarities of parts of these cases point out requirements and preferences to use SVSDM in different applications:

- online as well as offline situations in the sequence of the same work case
- mobile workers cooperating with fixed wired units
- monitoring different steps of the execution
- data transfer to and from mobile units
- a process of continuous development of a data package during the execution of the work task

All these factors are well handled by SVSDM and it is an optimal instrument to implement such cases. But also additional tools may be added to the system to help fulfil the predefined task as described in the "Use Case for a Mobile Medical Service", where the functionality of the SVSDM was enlarged by adding a WFMS to support the coordination of the activities.

SVSDM may be used in all cases where it is important to distribute data between clients who may stay offline for some time. SVSDM brings special features that make it possible that the client may work with data even when the online access is currently not possible. Therefore the system has a new way of data representation. The data is not saved into special folders on a predefined computer, but to a space designed to be part of each cooperating machine. The mechanism provided by the underlying Shared Virtual Space Manager (in my case CORSO) makes sure that the latest version of the content of the space is replicated onto all sites. Naturally, this also works for special cases where a central supervisor wants to be informed about all the events happening in the system. Therefore with each data transfer a log entry will be generated, which may be read by an administrator. This information may be used to generate detailed statistics or depictions of the execution.

VI. Evaluation

The development of a shared virtual space is not a simple task. May aim was to prove that SVSDM was designed in a way to be useful in many different use cases. The different potentials are shown in Chapter III Use Case for an Insurance Company, Chapter IV Use Case for a Mobile Medical Service, and Chapter V Other Use Cases, where a generalized applicability is outlined.

VI.1. Advantages

- **System independence**:
  SVSDM is implemented in Java, therefore it can be used on all systems supported by the Java Virtual Machine. The background software CORSO is already available for UNIX, Linux, z/OS and Windows. The portability of the program to other language is not considered of main importance, as Java is available for nearly every system, but if really needed this will not be very difficult as SVSDM together with CORSO solves the complexity for the communication, replication and synchronization tasks.

- **Recoverability**:
  The system was implemented with the usage of persistent communication objects so that system failures may be tolerated at any given moment. The previous state may be recovered after a system or a network failure. Special care was taken to avoid any form of inconsistency in data objects. This is reached by executing complex tasks using transactions. Their usage makes sure that the task is carried out as a whole or kept back completely.

- **Short development times**:
  SVSDM presents an easy framework for the quick implementation of a solution for the use cases mentioned above. The concept of shared virtual spaces helps a lot when work packages or collective data have to be exchanged.

- **Less source code**:
  Short development times imply that there is not much new source code to be written. The useful functions of the SVSDM give a straightforward approach for the implementation and help avoid complex functions.

- **Transactions**:
  The use of the transaction service integrated into CORSO gives SVSDM the possibility to assure that even complex actions put together in the semantic coherence of one transaction are carried out as a whole or rejected completely. CORSO supports long-running distributed and heterogeneous transactions. For example, the process of transferring a work package from the global space to the worker will be done in a transactional secure way to assure that the packet will arrive at the worker as a whole or remains at the global space (if any network or system failures have prevented the distribution).

- **Minimal Network Traffic**:
  The traffic of data on the network is minimized by using the local SVSDM cache, only if a packet is not available there or it is marked with *eager propagation* it will be fetched from the network.

- **Offline-Mode**:
  The offline state is supported fully transparent to all the users of the system. If the worker is not connected, she/he may work with the available data (buffered in the local space called LPTS) as if online. The worker even might create answers or produce new packets that will be transferred back as soon as a connection will be available.

- **Monitoring**:
  SVSDM provides a tool that can be used to overlook all the actions happening in the system. It may be used for statistic means to provide an analysis of the different steps occurring within the system and some information may be used to encourage the worker (e.g. most efficient worker, etc.).

VI.2. Future Work and Improvements

Some future developments were annotated while describing the use cases. Here is a list of some upcoming ideas:

- **Profile Management**:
  A profile management would be one of the greatest improvements. Such a system would be extraordinarily helpful in assigning the packets to their destination but as described in Chapter III.3.5 this is not a trivial task.

- **Distributed File system**:
  While implementing the use case solution for the insurance company, the idea was put to me whether it would be possible to implement a file system on the basis of SVSDM [Wert04]. The quick answer would be no, because SVSDM is not thought to act as a long time storage. But a system could be created that may profit from the pros of SVSDM and does not function as a storage. This system may be some kind of a synchronisation manager or a notification service for changed data. That system may help the insurance company to organize their data and keep it up to date. The clients may only have to remember one entrance point, not many shares.

- **Distributed Software installation**:
  This topic is very closely related to the file system, the data of the new software will be stored as long as it has not been installed onto all target systema. It is thought to be some kind of update service that keeps the software of all connected systems up to date.

- **Monitoring more flexible**:
  An included monitoring function could be improved and extended to a more expressive tool. It should also offer built in statistic analysis or an interface to a common statistic program.

- **Mobile Backups**:
  SVSDM could also be used as a mobile backup system. The contents of selected data folders might be copied automatically to a central storage as soon as any changes occur. Many well known companies try to create solutions where the data of the mobile workers are saved centrally. The task is to give a certain security to recover all data in case of any disaster that may happen. In order to prevent the loss of data, what would mean a lot of work.

VII. Conclusion

This thesis shows how the SVSDM – Shared Virtual Distribution Manager based on the space based computing paradigm (Chapter II.4, page 24) may be used in realistic use cases.

The first Use Case for an Insurance Company (Chapter III, page: 40) describes a real-life solution developed together with a big German insurance company. They were looking for a new way of coordinating and overlooking their mobile workers. The prototype implemented fulfils the requirements determined together with the IT department of the company. This prototype was presented at a meeting and is currently under investigation by the responsible executives. It is planned to extend the presented prototype according to the needs of the company and finally to use it in real life situations.

The Use Case for a Mobile Medical Service (Chapter IV, page: 66) and the Use Case for a Telecom Service Provider (Chapter V.1, page: 72) describe other real-life situations were SVSDM could be of great help. Both scenarios are depicted and a possible solution using the SVSDM is presented. For the medical service as a second tool a WFMS – WorkFlow Management System is added as it will ensure that all activities are executed in the correct order.

Finally the Generic Use Case (Chapter V.2, page: 74) tries to define generic requirements that may present a scenario suitable for the implementation of SVSDM.

The use cases depict how the SVSDM may be used in real-life situations. Today's business asks for mobile solutions that enable an employee to follow the customers instead of forcing them into the company's offices. Therefore mobile solutions which are able to use the central database of a business are of great importance.

The prototype could show that this can easily be achieved by using the SVSDM. Therefore SVSDM plays out its strength as it implements a distribution algorithm that is capable of managing online/offline situation. This is achieved by the space based computing paradigm and is supported by a transaction and notification service.

The two other use cases which are not implemented for quite different purposes promise to serve the users better than common solutions. The discussion of requirements for the Mobile Medical Service showed the feasibility of the joint use of the SVSDM together with a Workflow Management System.

The generic use case finally showed in general that the SVSDM framework can minimise the complexity of mobile IT solutions by offering a widespread functionality for today's businesses.

Acronyms

BPEL	Business Process Execution Language
BPEL4J	BPEL for Java
CERN	Conseil Européen pour la Recherche Nucléaire (French for European Organization for Nuclear Research)
CORSO	CoORdinated Shared Objects
CPU	Central Processing Unit
DB	DataBase
FPD	Flat Panel Display
FTP	File Transfer Protocol
GDV	Gesamtverband der Deutschen Versicherungswirtschaft (GDV), (German for Association of German Insurance Companies)
GPS	Global Positioning System
GPTS	Global Persistent Temporary Storage
GUI	Graphical User Interface
HMD	Head Mounted Display
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
ID	IDentification
IMAP	Internet Message Access Protocol

IP address	Internet Protocol address
IT	Information Technology
LAN	Local Area Network
LPTS	Local Persistent Temporary Storage
MP3	MPEG-1 Audio Layer 3, compression format for audio files
P2P	Peer to Peer (Peer 2 Peer)
PCMCIA	Personal Computer Memory Card International Association
PGP	Pretty Good Privacy
POP3	Post Office Protocol version 3
SVSDM	Shared Virtual Space Distribution Manager
TCP/IP	Transmission Control Protocol/Internet Protocol is the name for the Internet protocol suite
TTL	Time-To-Live
UDDI	Universal Description, Discovery, and Integration
VSM	Virtual Space Manager
Web	word wide Web
WFMS	WorkFlow Management System
WSDL	Web Service Description Language
WWW	Word Wide Web
XML	eXtensible Mark-up Language

References

Printed material

[Andr03] Andrews, Tony et al.: *Business Process Execution Language for Web Services Version 1.1*(BPEL4WS Specification), 5th of Mai 2003, downloaded the 20th of October 2003 from http://www.ibm.com/developerworks/webservices/library/ws-bpel/.

[Call97] Calladine, J.: *BT Middleware – software as infrastructure.* BT Technol J Vol. 15 No. 1 January 1997.

[Dust03] Dustdar, Schahram; Gall, Harald; Hauswirth Manfred: *Software-Architekturen für Verteilte Systeme.* Springer-Verlag, Berlin Heidelberg, 2003.

[Free99] Freeman, Eric; et al.: *JavaSpaces™ Principles, Patterns, and Practice.* Addison Wesley, 15th of June 1999.

[Gart01] Gartner Consulting: *The emergence of distributed Content Management and peer-to-peer content networks.* Engagement #010022501, Gartner Group, 2001.

[Graf03] Grafl, Andrea: *Neue Trends und Standards für WebServices und Peer-to-Peer am Beispiel von JXTA* (Diplomarbeit). Technische Universität Wien, 2003.

[Juri05] Juric, Matjaz B.: *A Hands-on Introduction to BPEL*, downloaded the 9th of August 2005 from http://www.oracle.com/technology/pub/articles/matjaz_bpel1.html

[Kanh02] Kanhäuser, Christian: *Peer-to-Peer Lösungen für Pocket PCs* (Diplomarbeit). Technische Universität Wien, 2002.

[Kühn94] Kühn, eva: *Fault-Tolerance for Communicating Multidatabase Transactions, In: Proceedings of the 27th Hawaii International Conference on System Sciences (HICSS), ACM, IEEE*. Wailea, Maui, Hawaii, January 4 – 7 1994.

[Kühn98] e. Kühn, G. Noicka: *Post-Client/Server Coordination Tools, In: Coordination for Collaborative Applications*. (Wolfram Cohen, Gustaf Neuman /eds.), Springer Series Lecture Notes in Computer Science, 1998.

[Kühn01] Kühn, eva: *Virtual Shared Memory for Distributed Architecture*. Nova Science Publishers, 2001.

[Mahm04] Mahmoud, Wuasay H.: *Middleware for Communications*. John Wiley & Sons, Ltd, 2004.

[Mina01] Minar, Nelson; et al.: *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly, February 2001.

[Mord05] Mordinyi, Richard: *Shared Virtual Space Distribution Manager –SVSDM– Design and Implementation* (Diplomarbeit). Technische Universität Wien, 2005.

[Orfa99] Orfali, Robert; Harkey, Dan; Edwards Jeri: *Client/Server Survival Guide, Third Edition*. John Wiley & Sons, Inc, 1999.

[Rech99] Rechenberg, Peter; Pomberger, Gustav; et al.: *Informatik-Handbuch*, 2., aktualisierte und erweiterte Auflage. Carl Hanser Verlag, München Wien, 1999.

[Riem03] Riemer, Johannes: *Peer-to-peer groupware mit CORSO* (Diplomarbeit). Technische Universität Wien, 2003.

[Stuc05] Stuckenschmidt, Heiner; Frank van Harmelen: *Information Sharing the Semantic Web*. Springer-Verlag, Berlin Heidelberg, 2005.

[Sutt01] Sutton, Stanley M.: *Middleware Selection*. Springer-Verlag, Berlin Heidelberg, 2001.

[Tane02] Tanenbaum, Andrew S.: *Distributed systems*. Upper Saddle River, NJ: Prentice Hall, 2002.

[Vino04] Vinoski, Steve: *An Overview of Middleware*. Springer-Verlag, Berlin Heidelberg, 2004.

[Wert04] Werthner, Georg: *Entwurf und Implementierung eines verteilten, fehlertoleranten und ausfallssicheren Dateisystems mit Corso* (Diplomarbeit). Technische Universität Wien, 2004.

Online material

[www01] Various definitions, downloaded in between April and August 2005 from http://en.wikipedia.org/wiki/.

[www02] *Xybernaut Products*, downloaded the 11th of April 2005 from http://www.xybernaut.de/produkte/ger/d_ma4.shtml.

[www03] *Definition of Middleware*, downloaded the 12th of April 2005 from http://philip.greenspun.com/seia/glossary/.

[www04] Definition for GDV, downloaded the 11th of April 2005 from http://de.wikipedia.org/wiki/.

[www05] *Prisma*, downloaded the 13th of April 2005 from http://www.silverstroke.de/prisma/.

[www06] *Business processes: Understanding BPEL4WS, Part 1* (published the 1st of August 2002), downloaded the 3rd of November 2003 from http://www.ibm.com/developerworks/webservices/library/ws-bpelcol1/.

[www07] Kühn, eva: *SBC-GRID Initiative* (last updated January 2005), downloaded the 15th of August 2005 from http://www.complang.tuwien.ac.at/eva/Research/researchSBC-GRID.html.

[www08] Kurschl, Werner: *Space-Based versus Message-Passing Communication A Comparison* (published February 2004), downloaded the 28th of August 2005 from http://webster.fh-hagenberg.at/staff/kurschl/pubs/TR.2004.01.SpaceBasedvsMessagePassing.pdf.

[www09] Tidwell, Doug: *Web services: the Web's next revolution* (published the 29th of November 2000), downloaded the 6th of September 2005 from http://www.ibm.com/developerworks/edu/ws-dw-wsbasics-i.html.

[www10] Juric, Matjaz: *A Hands-on Introduction to BPEL*, downloaded 9th of August 2005 from http://www.oracle.com/technology/pub/articles/Matjaz_bpel1.html.

[www11] Steindl, Christoph: *Agile Softwareentwicklung*, downloaded 14th of October 2005 from http://www.ssw.uni-linz.ac.at/Services/Seminars/AgileSoftwareentwicklung/

Lists

Figures

Figure 1: Web Service [www01] 17
Figure 2: BPEL Process [www06] 18
Figure 3: SBC-GRID Architectures [www07] 24
Figure 4: SVSDM Functions 25
Figure 5: Actual Setting 41
Figure 6: Data-Flow for the Prisma System [www05] 42
Figure 7: Scenario 45
Figure 8: Proposed solution for the insurance company 50
Figure 9: Producer Window 53
Figure 10: Consumer Window 57
Figure 11: Monitor Window 60
Figure 12: Workflow of Mobile Medical Service 64
Figure 13: Xybernaut Wearable Computer MA TC [www02] 90
Figure 14: Xybernaut Atigo T Wireless Display with stylus [www02] 91
Figure 15: Test Environment 93

Tables

Table 1: BPEL primitive activities 19

Table 2: BPEL structure activities 19

Table 3: Layers of the SBC-Grid Architecture 23

Table 4: List of Functions provided by SVSDM 26

Examples

Example 1: Example for an empty BPEL Process 20

Example 2: Example for a BPEL Process (exzerpts) 21

Example 3: Call to create new GPTS 29

Example 4: Call to create new LPTS 31

Example 5: Call to create new Notification Board 33

Example 6: Call to create new Life Beat Checker 34

Example 7: Call to create new Life Beat Checker 35

Example 8: Call to create new Packet 36

Example 9: Call to create new Restriction 37

Example 10: Example for abakus.ini 49

Example 11: Example for producer.ini 54

Example 12: Producer Code 55

Example 13: FolderThread Code 56

Example 14: Example for consumer.ini 58

Example 15: Consumer Code 59

Example 16: Example for monitor.ini 61

Example 17: Monitor Code 61

Example 18: DeployBpel 89

Appendix

Integration of BPEL4J into an automated environment

The task was to find a way to integrate the BPEL4J engine into an automated environment without changing the BPEL4J engine itself. Automated environment means that all manual steps of submitting a web formula are simulated by the software. The BPEL4J engine is a Java implementation of the BPEL standard. It has to be deployed on an application server such as an Apache Tomcat. After that the engine is available through a web interface. In the first step a bpel-file (the specification of the workflow, see Chapter II.2 for more information) and the related wsdl-file (the description of the service) have to be specified. If the business process uses external web services a second step will be necessary where one will have to define a wsdl-file for each service. After this procedure the business process (described by the bpel-file) is deployed and ready to be executed.

To achieve that goal the messages that are exchanged between the web browser and the BPEL4J engine had to be analysed. This way the key factors in that communication could be found out. Then a tool had to be found to create those messages. A toolkit provided by Apache called http-client solves that need. This is a library for Java that provides a lot of useful functions when communicating with a web server. This library creates the answers to the form and also parses the answers sent back from the server.

Finally an automated solution to a task that was originally intended to be done manually was created without having to change any detail in the BPEL4J engine:

Example 18 shows in pseudo code how this solution was implemented. There are two independent functions implemented. The deployment process may in some special cases consist of two parts. The first function `deployService(zipFile)` (lines 3-47) handles the first step which is necessary for all deployments. The second function `deployServicePart2(zipFile)` (lines 49-66) treats the second part of the process if necessary.

The function `deployService(zipFile)` takes a ZIP-File, which holds two files needed for the deployment process a bpel-file – the actual workflow description (see Chapter II.2 BPEL) and a wsdl-file – the service description (line 4). To start the deployment first the URL is created (line 7). It is composed from the `HOST`, `PORT` where the deployment server runs and a special prefix (`DESCRI`), which is used to connect with the service. Then the original cookies are loaded (line 8), because they will be needed for each request sent to the server to show that they have a context. The request page is created (lines 8-12) with the bpel-File and wsdl-File and the request is posted (line 13) to the deployment server. Then the answer is evaluated. The first two cases tell that there was either an open deployment process (line 16) or the workflow had been deployed earlier (line 26). In both cases the deployment process is restarted after either the older process was cancelled (lines 18-23) or the old workflow was undeployed (line 31). The other two cases tell either that the entire deployment went well (line 42) or that a second step is needed (line 34), for the uploading of some more wsdl-Files.

The deployment of the second part will start with the construction of the URL again (line 53). Then the files are uploaded (lines 55-59). Afterwards the request is posted and the deployment is finished.

```
DeployBpel {
// deploy a service on BPEL-Engine [first step]
  public int deployService(zipFile) throws Exception {
    File wsdlFile = new File(zipFile[0]); File bpelFile = new File(zipFile[1]);
    //try deploying until task is fulfilled
    while (true) {
      String targetURL = "http://" + HOST + ":" + PORT + DESCRI;
      Cookie [] cookies = getCookies();
      MultipartPostMethod filePost = new MultipartPostMethod(targetURL);
      filePost.addParameter("wsdlFile", wsdlFile.getName(), wsdlFile);
      filePost.addParameter("bpelFile", bpelFile.getName(), bpelFile);
      filePost.addParameter("step", "Continue Deployment");
      postBodyStr=client.postHtml(filePost, cookies);
      //check state of deployment
      switch(postBodyStr.getstatus())
      case("process of being") {
        //deployment open, abort old one then restart
        MultipartPostMethod cancelOpen = new MultipartPostMethod(targetURL);
        cancelOpen.addParameter("aborteddeployment", "Yes");
        cancelOpen.addParameter("flowid",getID(postBodyStr));
        client.setConnectionTimeout(5000);
        client.getState().addCookies(cookies);
        postBodyStr=client.postHtml(cancelOpen);
        continue;
      }
      case("already deployed") {
        //already deployed, undeploy old one, and restart
        // id from Process to be deleted
        String delID = getID(postBodyStr);
        //undeploy, get special undeploy page with ID of process to undeploy
        client.getHtmlFrom(UNDEPLOY+"?id="+delID);
        continue;
      }
      case("PartnerLink Identification") {
        //deployment fine, but step 2 necessary, save information for step 2
        saveID4step2 = getID(postBodyStr);
        cookies4step2 = cookies;
        // find PartnerLinks
        partnerLinks4step2 = getPartnerLinks(postBodyStr);
        return 1; //success, but step 2 is expected
      }
      case("deployed:") {
        //deployment fine, no step 2
        return 0; //success, no step 2
      }
    } //while
  } //deployService
// deploy a service on BPEL-Engine [second step]
  public int deployServicePart2(zipFile) throws Exception {
    if (saveID4step2 == "") {
      return 1; //no step 1, was processed; or step 2 not necessary
    }
    String targetURL = "http://" + HOST + ":" + PORT + DESCRI;
    //upload files
    MultipartPostMethod filePost = new MultipartPostMethod(targetURL);
    for (int i=0; i<zipFile.length; i++) {
      filePost.addParameter(partnerLinks4step2[i], zipFile[i].getName(),
                            zipFile[i]);
    }
    filePost.addParameter("step", "Start Serving the Process");
    filePost.addParameter("flowid",saveID4step2);
    client.postHtml(filePost, cookies4step2);
    //reset ID & cookies
    saveID4step2 = ""; cookies4step2 = null; partnerLinks4step2 = null;
    return 0; //success
  } //deployServicePart2
} //DeployBPEL
```

Example 18: DeployBpel

Mobile Computing solutions by Xybernaut

This chapter contains a description of the mobile hardware that was used for the demonstration of the Use Case for an Insurance Company (see Chapter III.3.4).

Xybernaut

The mobile devices used to demonstrate the work are so-called wearable computers. They were provided by Xybernaut [www02].

Wearable computing unit MA TC

The company provided three entities of the type MA TC. This product was designed to be carried in a specially manufactured belt.

Figure 13: Xybernaut Wearable Computer MA TC [www02]

Figure 13 shows the parts that can be used together to form a fully functional computer. The central processing unit module (CPU Modul) (picture 3) can be used either with a head mounted display (HMD) (picture 1) that has an overhead display, headphone, and a microphone, or with a flat panel display (FPD) (picture 2) with touch screen functionality including a speaker that was specially designed to be robust and very bright to recognize the contents in every light condition. The last picture shows the battery pack. The processor unit is also equipped with a battery in order to buffer some energy to make the hot swapping of battery packs possible.

Some facts about the parts, taken from the data sheet:

CPU module:

Magnesium alloy case, Processor: Intel Pentium III 400 MHz; Shock-mounted hard drive, 5 GB to 32 GB; Memory: 64 MB to 320 MB; 2 type II PCMCIA card slots; ESS Maestro Pro sound card; CT 69030 video chip 4 MB, SXGA and LVDS, SVIDEO; 2 COM and 2 USB interfaces; Bi-directional EPP (parallel port); Internal battery (up to 1h operation time); Measures: 18.7 x 6.3 x 11.7 cm; Weight approx. 900 g

Head-Mounted-Display (HMD):

Weight: approx. 470 g; 640 x 480 colour VGA monocular, left- or right-side wearable, Over- or under-viewable; microphone and ear-piece speaker; optional integrated miniature video camera.

Flat-Panel Display (FPD):

All-light readable displays 6.4" „viewable", 640 x 480, resp. 800 x 600 colour VGA; Activation: voice, pen and touch screen.

Batteries:

Internal NiMH battery enables hot swapping; External Lithium-ion with 4-6 hr charge, weight 454 g, AC power adapter / battery charger with protective circuit.

Wireless display unit Atigo T

This hardware device is the newest development of the company (see Figure 14). This new technology is a very small computer where the processing unit is integrated into a screen that may be used with a stylus. The innovation is that a wireless LAN card is already integrated. However a drawback might be that the memory built in is very small, so running out of free space is possible.

Figure 14: Xybernaut Atigo T Wireless Display with stylus [www02]

Some technical details about the device, taken from the data sheet:

CPU module:

Transmeta Crusoe TM5800-1 GHz Processor, 256 MB SDRAM;

Flash Memory Options:

1 GB, 2 GB and 4 GB (with Windows XP Pro or Windows XP Embedded);
128 MB,256 MB,512 MB,1 GB,2 GB and 4 GB (with Linux Embedded,Linux);

Slots:

PC Card (CardBus Type II) CompactFlash Type II (CF-IO)

Ports:

1-USB 2.0

Audio:

Integrated 16-bitStereo Audio System, Stereo Speakers, Headset Jack (2.5 mm Cell Phone type with Microphone and Earphone)

Batteries:

Internal Rechargeable Lithium-Ion Battery; Optional Hot-swappable Clip-On Battery;
Optional High-Capacity Battery Pack;
AC Adapter (100-240V AC, 50-60Hz)

Facts:

Operation temperature: 0º~40ºC, Humidity: 0%~90%;
Unit size H x W x D: 200 mm x 240 mm x 18 mm;
Unit weight: (840 g), including battery;

Modifications

To use these units in the productive environments, some changes had to be made:

The wearable computing units had to be equipped with a wireless card module, as it will not be very useful to have a portable computer which has to be plugged onto a network cable. Therefore wireless card modules where plugged into the PCMCIA-slots. These modules and the wireless LAN support had to be configured manually as it was not possible to install the latest Microsoft Windows XP operating system, which would have had wireless networking support integrated. Because of this problem the Microsoft Windows NT system installed had to be adjusted to work with this module. This process enabled all units to work together and to exchange data through the wireless interface.

Mobile work was easier using the wireless display unit, as it was already equipped with a wireless module and the wireless support was given by the preinstalled Microsoft Windows XP Embedded operating system.

Finally an Access Point had to be found and integrated into the test environment to be able to connect the wireless units to the wired network.

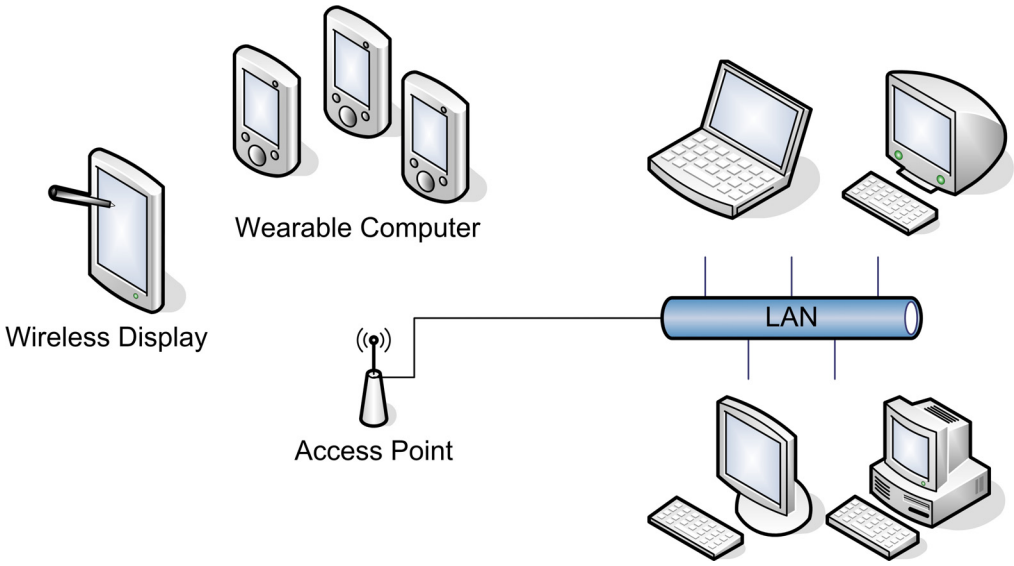

Figure 15: Test Environment