



Department of Electrical and Computer Engineering
Schulich School of Engineering

Project Report for **SENG609.22**
Agent-based Software Engineering (Fall 2005)

Instructor: Dr. B. H. Far

Agent-based Online Event Ticket System

Matthias Galster

October 25, 2005

Table of Contents

List of Figures	3
List of Tables	4
Abstract	5
1 Introduction.....	6
2 Overall System Specification.....	7
2.1 Problem Description	7
2.2 High Level Design Description	7
2.3 Assumptions.....	9
2.4 Requirements	9
2.5 Wish List of Not Implemented Features (Future Requirements).....	10
3 Top Level System Design.....	11
3.1 Roles and Actors	11
3.2 Architecture Description.....	13
3.3 Agent Description	15
4 Technological Overview	22
4.1 Publishing Service	22
4.2 Database and Programming Language	22
5 Detailed Design.....	24
5.1 Use Cases	24
5.2 Inter-Agent Communication Specification	30
5.3 Web Service Communication	33
5.4 Data Specification.....	34
6 Conclusion	36
References.....	37

List of Figures

Figure 1: Informal structure of the OTS.	8
Figure 2: <i>Tropos</i> actor diagram for the online ticket system.	12
Figure 3: Refined <i>Tropos</i> actor diagram for the actor <i>OTS</i>	13
Figure 4: Architectural structure of the OTS.	14
Figure 5: Internal structure of BookingAgent.	16
Figure 6: Internal structure of EDatabaseAgent.	17
Figure 7: Internal structure of PDatabaseAgent.	18
Figure 8: Internal structure of TicketFindingAgent.	19
Figure 9: Internal structure of PaymentAgent.	20
Figure 10: Internal structure of TicketManagementAgent.	21
Figure 11: Use case BookingAgent.	24
Figure 12: Use case TicketFindingAgent.	26
Figure 13: Use case PaymentAgent.	27
Figure 14: Use case TicketManagementAgent.	28
Figure 15: Use case PDatabaseAgent.	29
Figure 16: Use case EDatabaseAgent.	30

List of Tables

Table 1: Use case description BookingAgent.	25
Table 2: Use case description TicketFindingAgent.	27
Table 3: Use case description PaymentAgent.....	28
Table 4: Use case description TicketManagementAgent.....	29
Table 5: Use case description PDatabaseAgent.....	29
Table 6: Use case description EDatabaseAgent.....	30
Table 7: Data description of event database.	34
Table 8: Data description of profile database.	35

Abstract

The following paper describes an internet-based multi-agent Online Ticket Service (OTS). The system allows the customer to buy tickets for events (concerts, musicals, etc.) online, whenever and wherever she/he wants. The ticket system has access to events in an internal data base as well as direct access to events that are made available by event providers using their own external web services. Several agents are introduced to support this task. The paper will present the business case in detail, the architectural description, a technological overview, and design details.

1 Introduction

The entertainment industry is one to the most profitable sectors in the business world. People always spent money for being entertained, and so will they in the future. As experience shows, people are mainly attracted by live events like concerts, operas, sport games, or revue shows. It gives people a great and unique entertainment experience. The only problem is how people get to know about these events, and then, in case they know, how they can attend these events, i.e. how can they get tickets (even for events that are far away from the customer's place). This problem used to be solved by ticket stores. Unfortunately in most cases these are restricted by time, location, and – most important – by availability of tickets (available tickets are mostly limited to events taking place in a certain local area where a ticket store is located). Because of that reason the following document will describe the implementation details of an Online Ticket Service (OTS). It is intended to work as a web-based multi-agent application. The system will allow the user to buy tickets online for events that are either stored in the system's own database or available using web services offered directly by the event provider (either other booking agencies or the presenter directly, like e.g. the musical theatre or the artist itself), but accesses by the OTS. Bookings can be made independent of time and location.

The structure of the document will be as follows: In the second chapter we will give an overview of the system specification which will be refined in the system design that is presented in the third part. A technological overview follows in the fourth chapter and the design details of the system are introduced in chapter 5. Additionally we will draw a conclusion at the end of this paper.

We will use the following terminology throughout the document:

<i>Client</i>	Entity outside the system that has access to the system.
<i>Customer</i>	Person that applies the application to purchase tickets.
<i>Feature</i>	Requirement.
<i>Requirement</i>	Feature that should be fulfilled by the system.
<i>User</i>	Person that uses the application, either as customer or administrator.

2 Overall System Specification

2.1 Problem Description

Like in many other industries and commercial branches customers in entertainment industry ask for services that are more and more available independent from time and location. People want to buy tickets for events regardless of the time, whenever they want, and wherever they are. Offering services which are independent of time and location is one of the main problems, not only in the event industry.

Customers are using the internet as a medium to purchase many things for which they used to go to physical stores, and so do they for event tickets. When searching for different kinds of events (like musicals, concerts, operas) customers need a ticket dealer to show and sell them what is available according to their wishes, or they consult specific event providers, such as websites of music groups, websites of theatres, etc. Here, intelligent agents have a great potential in helping the customer to get what she/he wants, whenever she/he wants.

There are various reasons for us to build OTS as an agent-based system. Agent-based systems provide expandability and scalability as well as support for heterogeneous mobile structures. Additionally, our OTS system is a non-monolithic program model (what is supported by agent-based paradigms) and the agents will be inhabited by other processes. More reasons are:

- need for inference capability in OTS.
- reactive behavior of components in OTS.
- collaborative work, roles, responsibilities, and goal directness (autonomous single agents in OTS work together to achieve root goal, see *Tropos* models Figure 2 and Figure 3).
- entities in OTS can communicate and interact with other entities and artifacts in the outer environment (OTS uses external web services).
- OTS agents will be knowledgeable and able to manage the information on their own.
- OTS agents will be able to filter communication and managing dialogues that are defined with respect to their environment.

2.2 High Level Design Description

The proposed online ticket system is a multi-agent web-based system designed to purchase tickets for different events according to the preference of the customer and independent of time and location of the customer and the event. The usage shall be as simple as possible why we propose a simple web interface that can run on any web browser (on a personal PC, Smart Phone, etc.). No download of special client side agents is necessary. The system does not only offer events that are stored in the system's own

database but also has access to events which are provided by event providers (either booking agency or presenter) via their own web services.

The item sold by our system is one or more ticket(s) for a certain (or more) event(s). The OTS application is able to deal with different booking agencies, event presenters, etc. The application has to find the ticket the customer wants and proceed with the transactions related to the purchase of the ticket. Input to the event database of the system can either be made by the operator of the OTS or by external event providers.

The following figure (Figure 1) shows an informal description of the proposed software system.

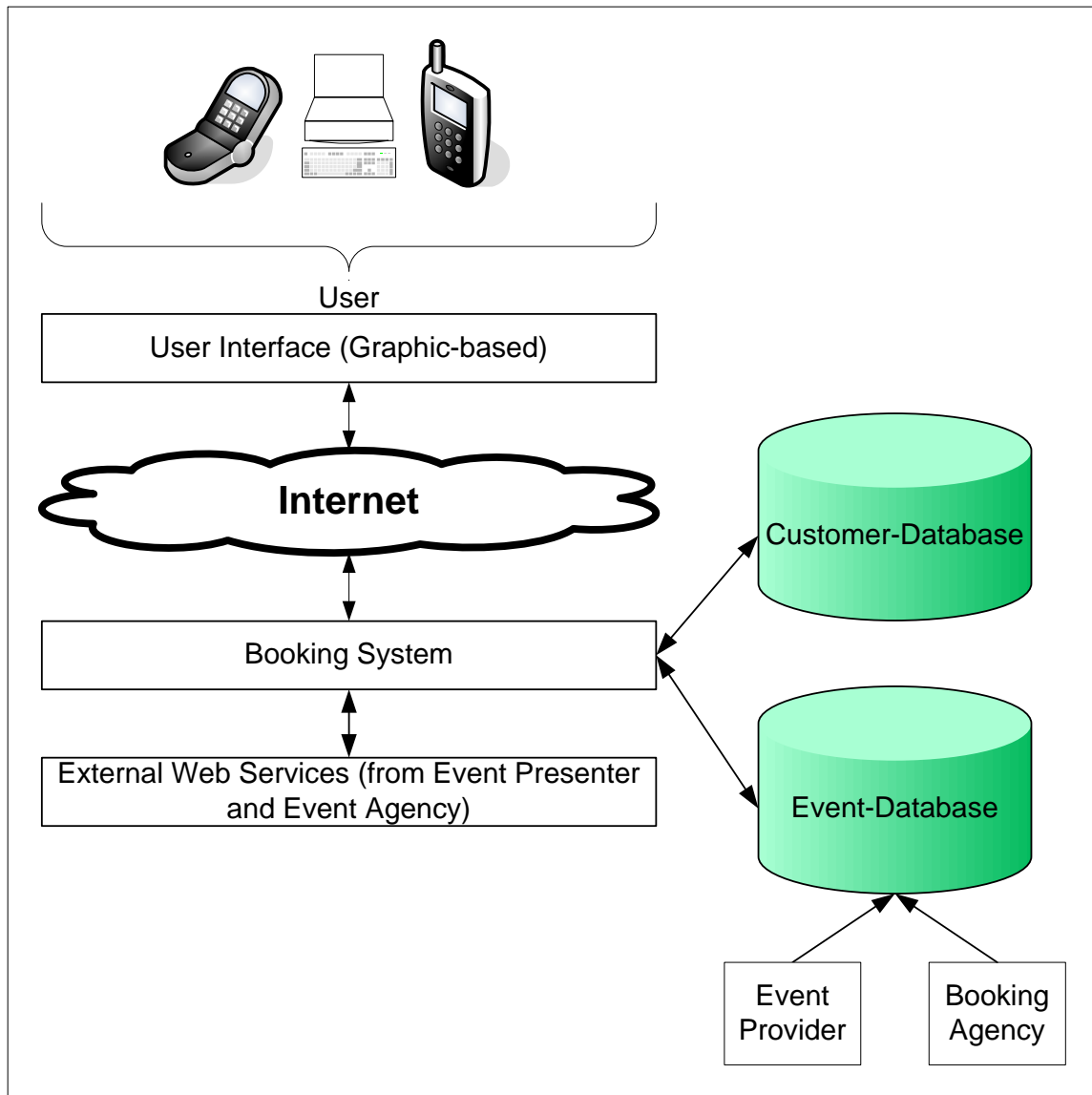


Figure 1: Informal structure of the OTS.

2.3 Assumptions

We will consider the following assumptions for the development of the OTS software system:

The system shall maintain a user profile (*Customer-Database* in Figure 1) for each user but that is not necessary. A user profile may be helpful for frequent customers to store customer related information (e.g. payment information, shipping information, preferences for customized offers and mail advertising, etc.). Preferences could be any type of event (pop concert, classical events, open air, etc.), locations, or days. In the remainder of this document the *Customer-Database* in Figure 1 will be addressed as *Profile-Database*.

The customer asks for events using a graphical user interface that can be used with every web browser what makes it available at every device with internet access (PC, Smart Phone, PDA, etc.). The interface captures the event information and the system retrieves events based on that information. For frequent users, profile information is also taken into consideration.

The customer can buy as many tickets he wants and available for one event. He also can buy tickets for different events at one time. The customer shall be able to buy tickets for events from all over the world.

The ticket booking system may have access to already existing infrastructures of music labels, other booking agencies, and event agencies. These web services must support the same Web Service Definition Language (WSDL) description, i.e. their input and output must have a standard syntax.

2.4 Requirements

The software system must fulfill the following requirements in order to meet the ultimate goal of customer satisfaction by providing a broad ticket service independent of time and location:

- R1. The OTS must provide event information and allow enquires.
- R2. The OTS must be able to deal with different event and booking agencies as well as direct event providers, local organizers, and presenters.
- R3. The OTS must be able to book the desired event upon the approval by the customer.
- R4. The OTS must be able to send a booking confirmation and payment confirmation to the customer after approval of booking and payment.
- R5. The OTS must allow the customer to create a user profile if desired by the customer.
- R6. When a user profile exists, the OTS shall be able to use customer information to process payment.

- R7. The OTS must allow entries to the system's database by the operator of OTS and by external event providers.
- R8. The OTS must allow appropriate handling of tickets when sold.
- R9. Tickets must be available for events at any location in the world.

2.5 Wish List of Not Implemented Features (Future Requirements)

The following features are not implemented in the first version of OTS but are planned to be added in future releases (to achieve a higher level of customer satisfaction):

- W1. When a user profile exists, the OTS shall be able to send a reminder of the booked event(s) to the customer one week prior of the event.
- W2. When a user profile exists, the OTS shall be able to provide a personal calendar for the customer to allow convenient planning.
- W3. If an event is canceled, the customer should be notified automatically.
- W4. Travel arrangements to the event shall be possible (e.g. booking a train, bus, hotel, etc.) by having appropriate external connections to travel planning systems (related extension to R9).
- W5. During the process of booking of a specific event the event item shall be held off from the inventory from the moment it is selected to prevent overbooking in the multi-user system and multiple accesses to one item.
- W6. When a user profile exists, the OTS shall be able to send customized offers of upcoming events to the customer.
- W7. The booking system shall allow higher level of security for payment and support *Paypal*.
- W8. Agents should use lookup services to locate existing web services, and their selection of web services should be based on trusted rating services.

3 Top Level System Design

This section will follow agent-based development methodologies and create the analysis and design documents (system architecture, role identification, agent description). We will apply the *Tropos* methodology for the design and analysis of the system. Details of *Tropos* can be found in [1]. The *Tropos* development process is characterized by certain features:

- design process is generic and focuses on a goal analysis for different actors.
- follows a non-deterministic concurrent algorithm.
- starts with actors with a list of related goals.
- analyzes root goals from the actor's perspective.
- generated sub goals are delegated to other actors.
- analysis is carried out concurrently with respect to each root goal.
- process may require the introduction of new actors.
- process ends when all goals have been dealt with to the satisfaction of the actors who want them.

The OTS should work as a layer of multi-agents. The service also uses other web services coming from one or more external providers (for events that are not available in the system's event database). They return information of their offered services to the agent based on client (OTS layer) input. This input may include a user profile that is processed by the OTS layer. The agents then try to get the result in a way that the customer gets the event he wants.

3.1 Roles and Actors

We use the *Tropos* approach to define the actors and top level goals of the system. Figure 2 shows the respective *Tropos* actor diagram which is created from the early requirements and results from the analysis of social and system actors and their goals and dependencies for the achievement of goals. Clouds describe softgoals (having no clear-cut definition or criteria), rounded rectangles depict hardgoals, and circles describe actors and roles.

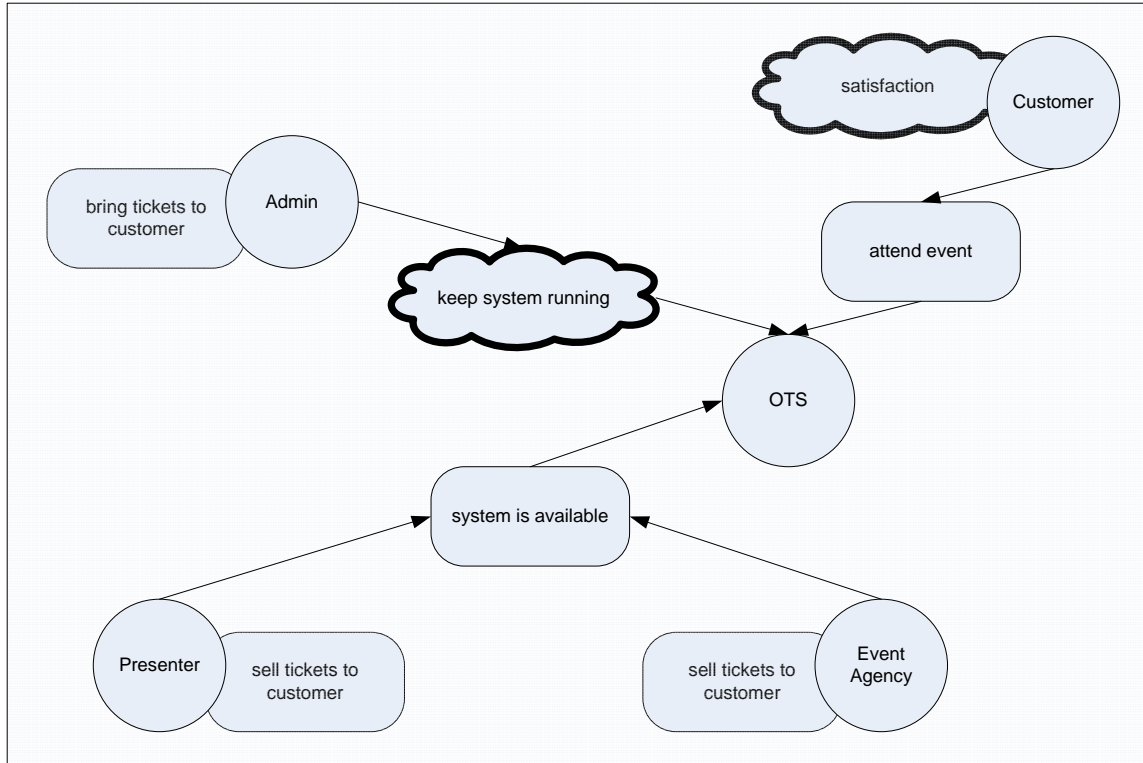


Figure 2: Tropos actor diagram for the online ticket system.

As it can be seen in the actor model (Figure 2) the following roles were identified for OTS:

Administrator. The administrator can enter event information into the event database. This might be an external event provider or the operator / provider of the OTS itself.

Customer. The customer selects an event (or more events), buys and pays for an event. In some cases the customer may create a profile in OTS.

Event Provider. An event provider provides events that can be purchased using the OTS. Event providers can be either booking agencies that have contracts with various artists or scheduled events or the event presenter directly, like opera houses, concert halls, or even the artist itself. When applying the *Tropos* approach this role is therefore refined to the roles **Presenter** and **Event Agency** (see Figure 2).

OTS. The system itself will be our main actor to fulfill all the requirements for the actors.

For further investigation we refined the actor *OTS* (the system itself) from Figure 2 to break the system down to the agent level (see Figure 3). This can also be seen as the architectural design level.

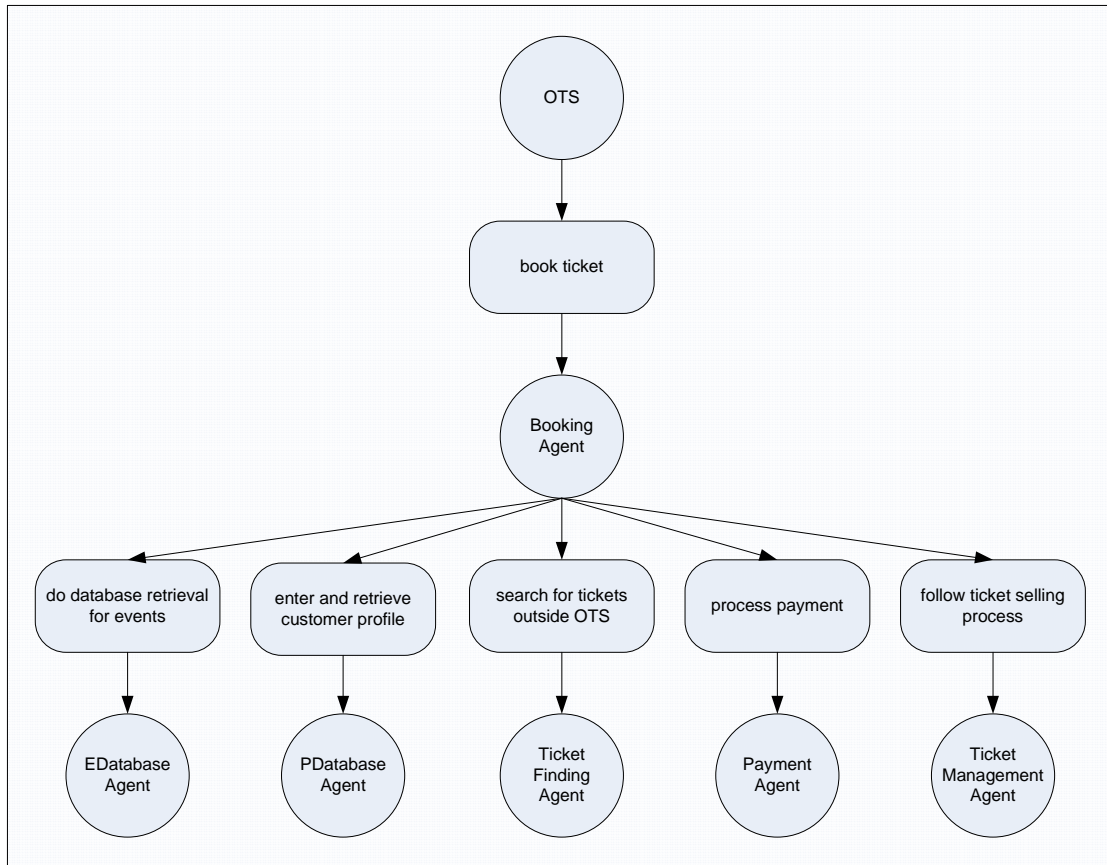


Figure 3: Refined Tropos actor diagram for the actor *OTS*.

Using this model we can derive our agents (see Figure 4).

3.2 Architecture Description

Figure 4 shows the architectural description of the OTS.

The system requires a user submitting a request. We have to differ between two kinds of clients / users:

1. A *system administrator* entering event information into the OTS event database. This is done using a web browser interface.
2. A *customer* submitting a request using an internet browser interface.

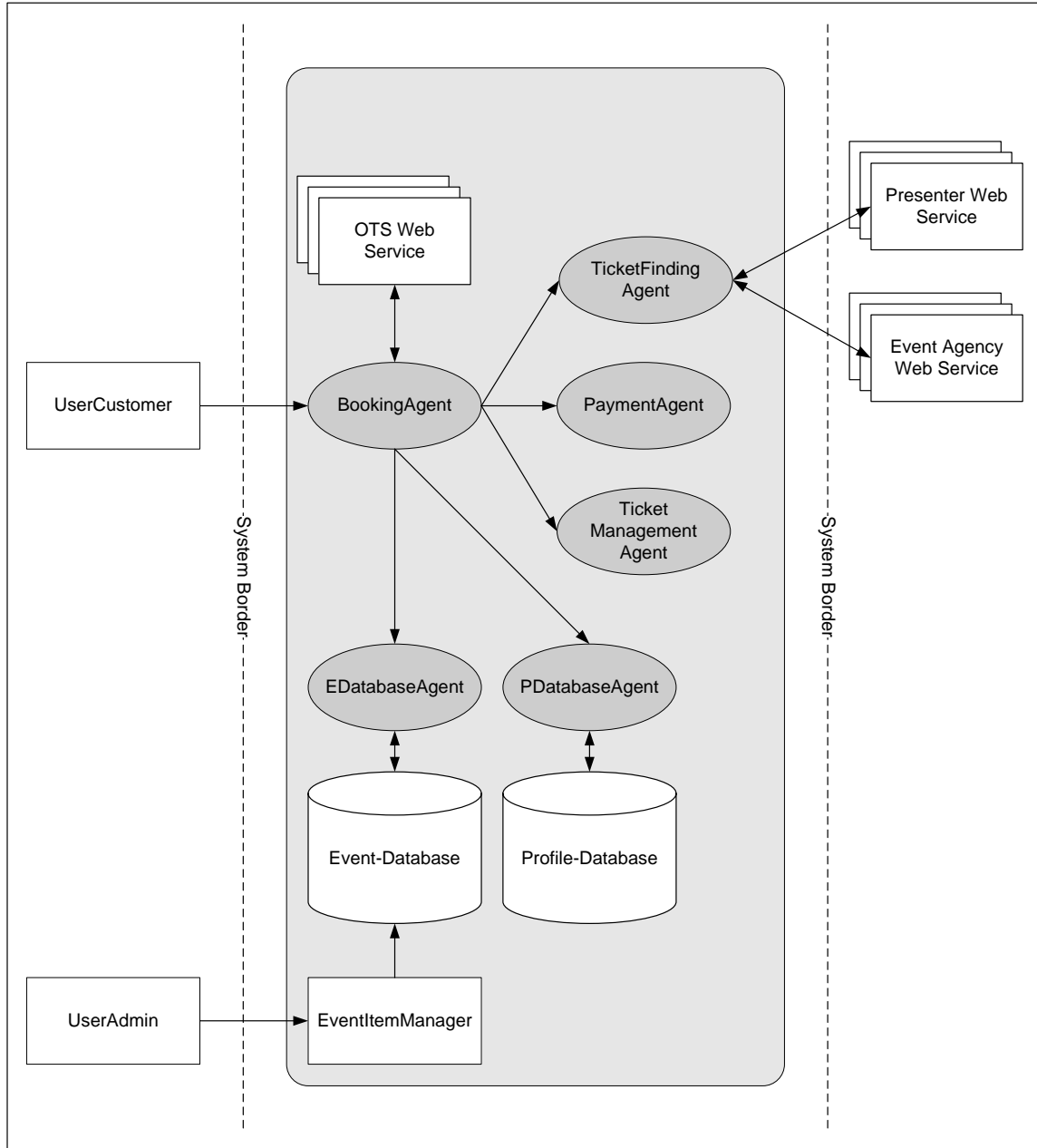


Figure 4: Architectural structure of the OTS.

The main input screen for the customer allows retrieving or browsing the event database according to several features, like date, location, kind of event, precise information of the event (name of artist, name of the operetta or musical, etc.).

The customer can create a profile if she/he wants. That allows the use of additional features: it eases the process of shipping and payment as related information is stored and can be reused for every transaction the customer does. Creating a profile has to be done by hand. Creating a profile is not necessary but useful for frequent customers.

The multi-agent system has to interact with the web services from external event providers if these providers support an own web service. Otherwise the information of event providers that do not support an own web service are entered into the system's event database by the system administrator. The role of the system administrator for this action can either be fulfilled by the overall administrator of the OTS operator or the event provider directly has the right to enter information in the event database of the OTS.

3.3 Agent Description

As visualized in Figure 3 and Figure 4, the system has basically six agents which will be described in the following more detailed.

BookingAgent. The BookingAgent receives the request sent by the user (customer) through the web interface. It handles the communication with all other agents. It can be seen as a central point of delegation for the whole system (also responsible for scheduling as a kind of facilitator). The BookingAgent requests products either from the internal database by forwarding the requests to the EDatabaseAgent or from web services provided by external event providers (by forwarding to TicketFindingAgent). It is also responsible for displaying the results on the GUI, also represented in the web interface. It sends requests to the agents to book selected items. Then it creates a memo that contains the information about the ticket booking and sends it to the TicketManagementAgent. Finally the BookingAgent has to trigger a search for the customer's profile to ease the payment process. If no profile is available, the customer has to enter payment information by hand for each transaction. Figure 5 shows the internal architecture of the BookingAgent.

Component description:

Listener. The Listener listens for any incoming requests from the user requesting for an event. It is the interface to the world outside the system.

Interpreter. Our transfer protocol standard is XML so the Interpreter has to parse and interpret all incoming XML messages (see chapter about the technological overview).

Processor. The Processor gets the parsed XML document from the Interpreter and runs the appropriate function.

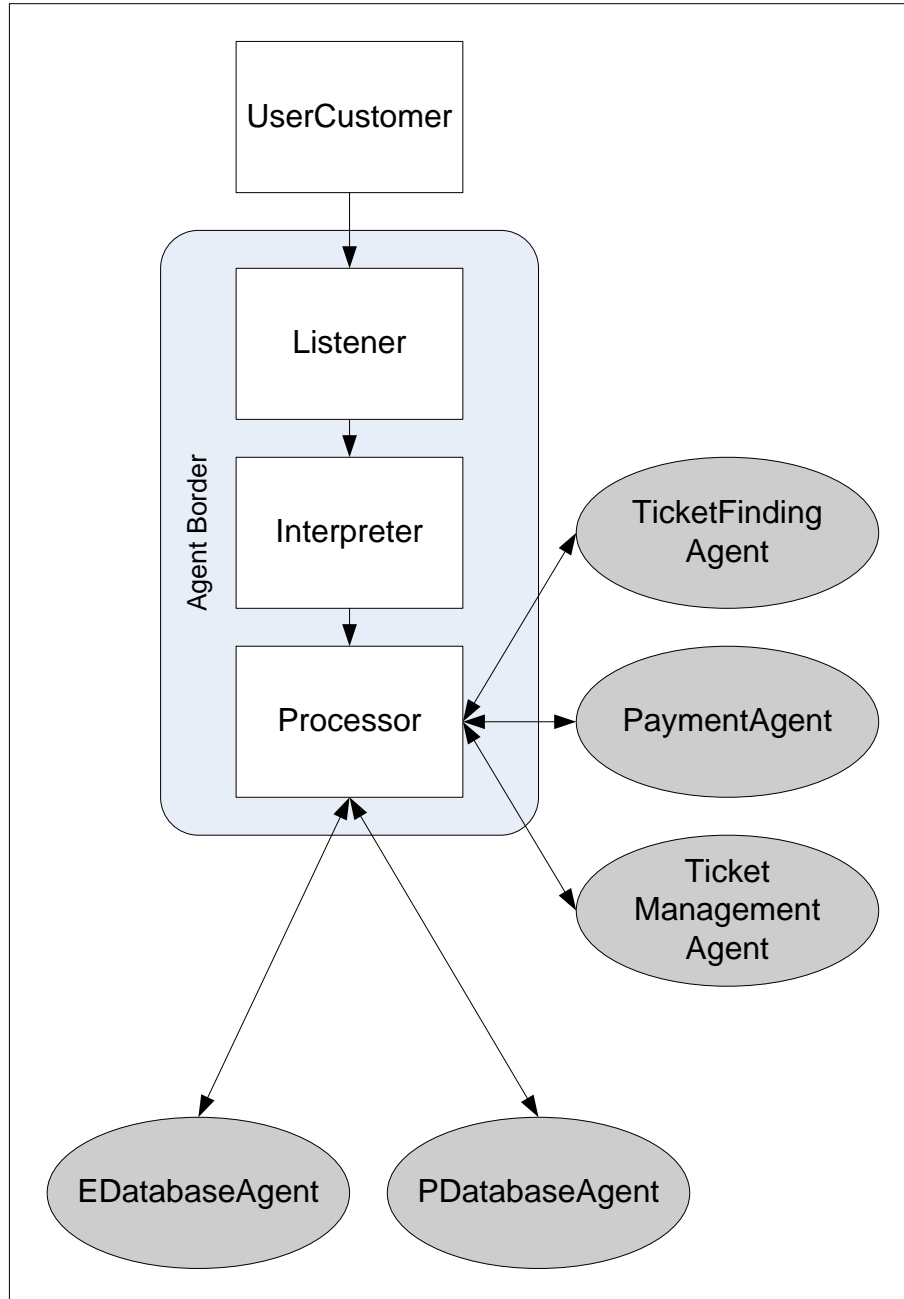


Figure 5: Internal structure of BookingAgent.

EDatabaseAgent. This agent (Figure 6) communicates with the BookingAgent. The EDatabaseAgent is responsible for managing all enquiries that are related to the event database. Requests are sent by the BookingAgent. The EDatabaseAgent returns the item(s) found for each request (event and number of tickets) if available. If no ticket(s) for a specific event is (are) available, a standard *no_success* notification is returned to the BookingAgent. If the number of requested tickets is larger than one and the number of tickets available is smaller than the number of tickets requested, the number of available ticktes is returned.

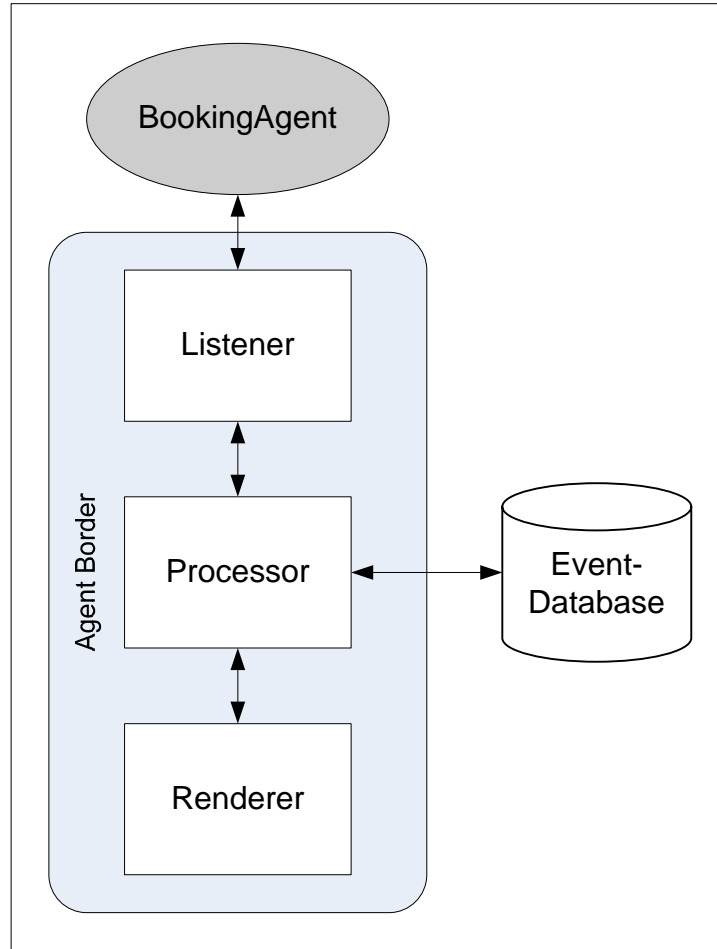


Figure 6: Internal structure of EDatabaseAgent.

Component description:

Listener. The Listener listens for any incoming requests from the BookingAgent.

Processor. The Processor gets the parsed XML document and sends a request to the event database.

Renderer. The Renderer is used to render the retrieved data before sending them back to the calling BookingAgent. The result from the event data base is basically an SQL statement.

PDatabaseAgent. The PDatabaseAgent (Figure 7) also communicates with the BookingAgent. It is responsible for handling all enquiries that are related to the profile database. It works similar to the EdatabaseAgent. Requests regarding customer profiles are sent by and to the BookingAgent. The agent returns the data fields requested by the BookingAgent. In case there are no information available for a certain customer, a standard *no_success* notification is returned.

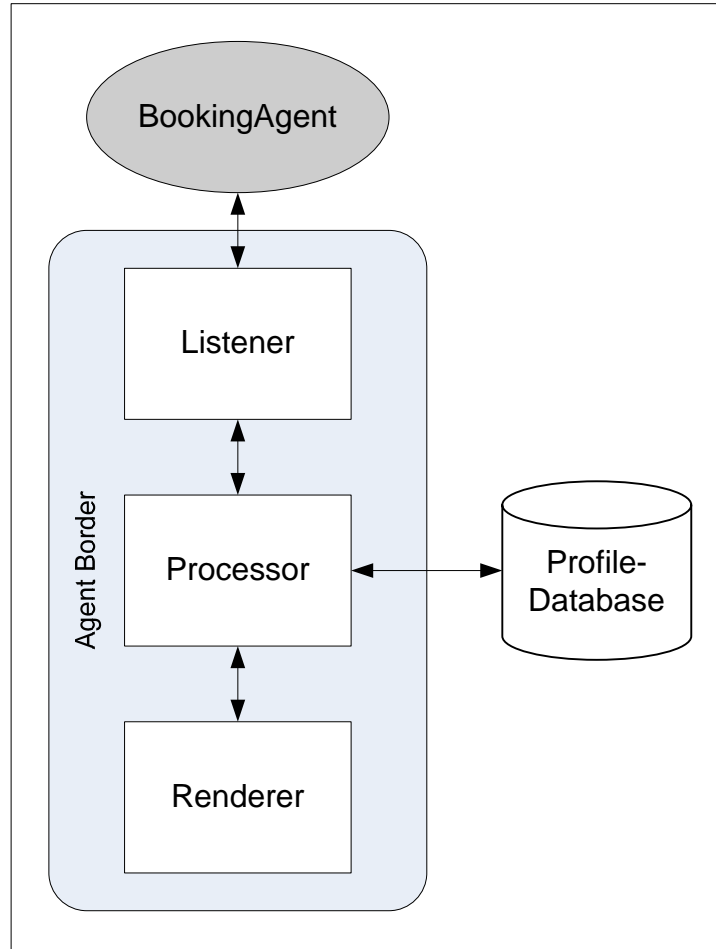


Figure 7: Internal structure of PDatabaseAgent.

Component description:

Listener. The Listener listens for any incoming requests from the BookingAgent.

Processor. The Processor gets the parsed XML document and sends a request to the profile database.

Renderer. The Renderer is used to render the data from the database before sending them back to the calling BookingAgent. The result of the profile database is basically an SQL statement.

TicketFindingAgent. The TicketFindingAgent (Figure 8) is responsible for finding appropriate tickets for the customer accessing external web services. For this action it communicates with the BookingAgent. The notification is sent by the BookingAgent when no appropriate tickets could be found in the system's own database. The requests are processed and then forwarded to the external web services. These web services return their result which is then rendered to a format that can be handled by the BookingAgent. The BookingAgent either receives information about available tickets or a standard *no_success* message.

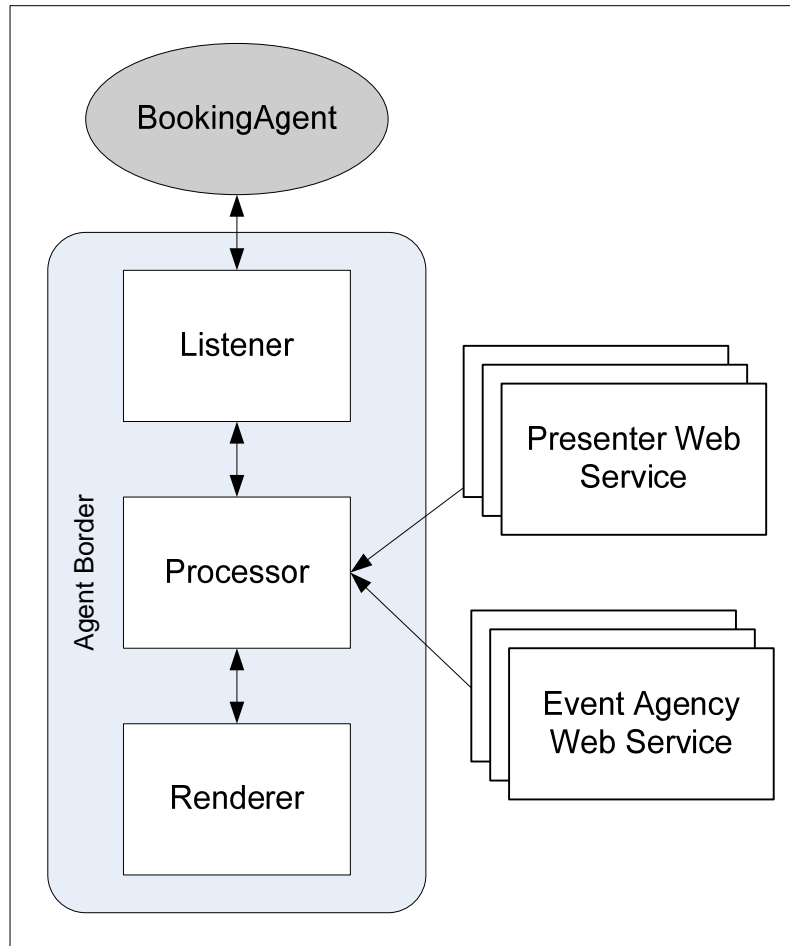


Figure 8: Internal structure of TicketFindingAgent.

Component description:

Listener. The Listener listens for any incoming requests from the BookingAgent.

Processor. The Processor gets the parsed XML document and calls the external web services.

Renderer. The Renderer is used to render the external data before sending them back to the calling BookingAgent.

PaymentAgent. This agent (Figure 9) communicates with the BookingAgent and is responsible for the payment process. Once a customer has decided and conformed that she/he wants to have a ticket or more different tickets for a certain event, the PaymentAgent is notified by the BookingAgent.

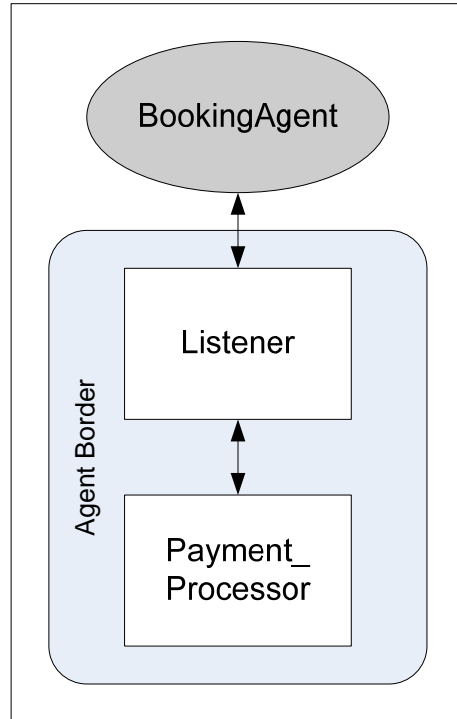


Figure 9: Internal structure of PaymentAgent.

Component description:

Listener. The Listener listens for any incoming payment requests from the BookingAgent.

Payment_Processor. The Processor gets the parsed XML document and executes the payment functions for the current transaction.

TicketManagementAgent. The TicketManagementAgent (Figure 10) is responsible for the shipment of the tickets and communicates with the BookingAgent. It also has to take care that the ordered tickets are booked out from the internal event database. It sends a notification to the customer informing that the payment has been accepted, tickets have been sent, and everything is in order.

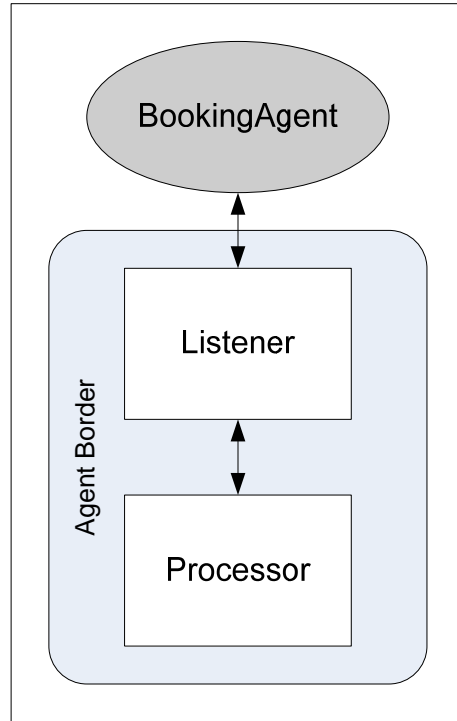


Figure 10: Internal structure of TicketManagementAgent.

Component description:

Listener. The Listener listens for any incoming requests from the BookingAgent.

Processor. The Processor gets the parsed XML document and executes the functions for the current transaction.

4 Technological Overview

The following part gives an overview of the technologies used to implement the OTS. As OTS is a web based service, web technologies and web services (for details see [2]) will be used to build the system.

4.1 Publishing Service

There are two categories of information that must be handled by OTS: *logical information* (description of the service to drive the ticket service what basically is the business information) and *technical information* (interface parameters of the logic, such exact protocol, inputs, outputs, encoding of messages used in the ticket service). The technical information forms a crucial part in machine-to-machine conversation.

To use a service it must be visible to the other users and describe its business. Therefore the web services must provide short descriptions of themselves on the web. To do this there are different ways. We decided to use the *JINI Discovery (java code discovery approach)*. In this approach each JINI service has a proxy object. Proxy objects allow the clients to use to interact with the service. A so called *look up service* maintains these objects. A JINI service publishes its proxy with the lookup service that other services can see it. The lookup service is a service so other services need to find its proxy to use. In JINI this is solved by building communities of JINI services. Services can periodically send messages over the network so that each service is aware of each other and so could assemble communities. As soon as a JINI service is found, the proxy of this service is downloaded to the client that requested for it. This client only needs to know how to interact with this proxy object. As these proxies are implemented in Java, the client needs a JVM (Java Virtual Machine) running. For services written in any other language a Java Surrogate is needed for JINI clients to communicate with these services [3].

Internal and external web services must support the same Web Service Definition Language (WSDL) description [4].

4.2 Database and Programming Language

The database will be an Oracle database and the programming language will be Java because its interoperability with the web technologies and the SOAP communication protocol [5]. Moreover, Java and Oracle support each other what makes it the perfect choice for our system. The inter-agent communication will be XML-based. The web services must follow a common WSDL, also defined in XML (see chapter 5 in the detailed design).

The JADE platform was chosen for the agent-based development because of its provided features that support our development:

- compliance with FIPA for agent-based interoperability.
- graphical tools supporting debugging, management and monitoring phase.
- support for message content creation and management including XML.
- support for Java, JSP, HTTP, IIOP.
- independent from underlying network and platform.

5 Detailed Design

The following chapter will give a detailed description of the components of the OTS. Use cases for all the agents will be presented as well as the communication details and the data specifications.

5.1 Use Cases

In the following we will present use cases for all agents.

BookingAgent.

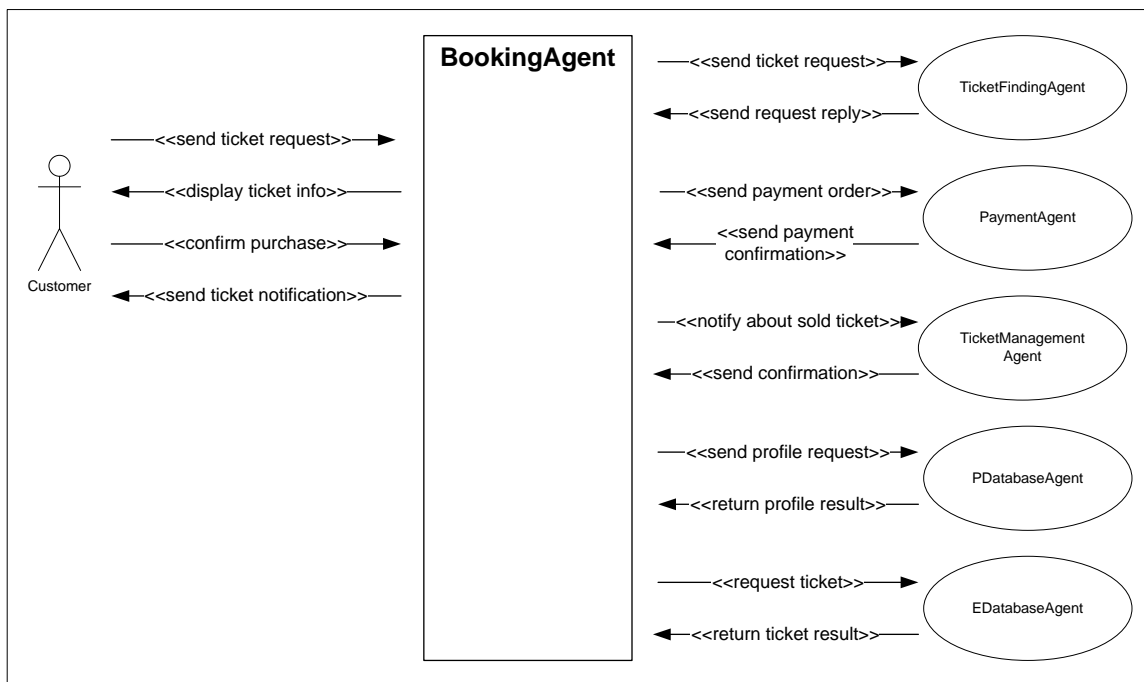


Figure 11: Use case BookingAgent.

A Brief Description	Customer actor requests and purchases an event ticket.
B Precondition	User profile is checked and loaded in case of existence.
C Postcondition	If purchasing process was successful the ticket(s) will be booked out of the system.
D Processing Step	
1	Customer sends a request for a certain event, at a certain date with a certain number of tickets.
2	BookingAgent receives request.
3	BookingAgent checks if a profile exists for this customer by sending a request to the PDatabaseAgent.

4	PDatabase Agent returns profile information; if there is no profile a standard <i>no_success</i> message is returned.	
5	BookingAgent forwards event request to the TicketFindingAgent and to the EDatabaseAgent.	
6	BookingAgent receives reply from TicketFindingAgent and EDatabaseAgent.	
7	BookingAgent shows results to the customer.	
8	Customer selects event or cancels action based on the list shown in the browser.	
9	BookingAgent notifies TicketManagementAgent about ticket purchase.	
10	TicketManagementAgent returns confirmation.	
11	BookingAgent notifies PaymentAgent to process payment.	
12	PaymentAgent returns confirmation.	
13	BookingAgent sends notification to PDatabaseAgent to update customer information in case a profile exists.	
E Exceptions		
1a	System is not accessible.	Error message is generated saying that system is not accessible. The use case is terminated.
7a	Customer cancels action.	Connection is closed, start page is shown.
F Relationships		
Initiating	Customer.	
Collaborating	TicketFindingAgent, PaymentAgent, TicketManagementAgent, EDatabaseAgent, PDatabaseAgent.	
G Data Requirements		
Data required for BookingAgent	(Event name OR event category OR event location OR event date) OR number of tickets.	

Table 1: Use case description BookingAgent.

TicketFindingAgent.

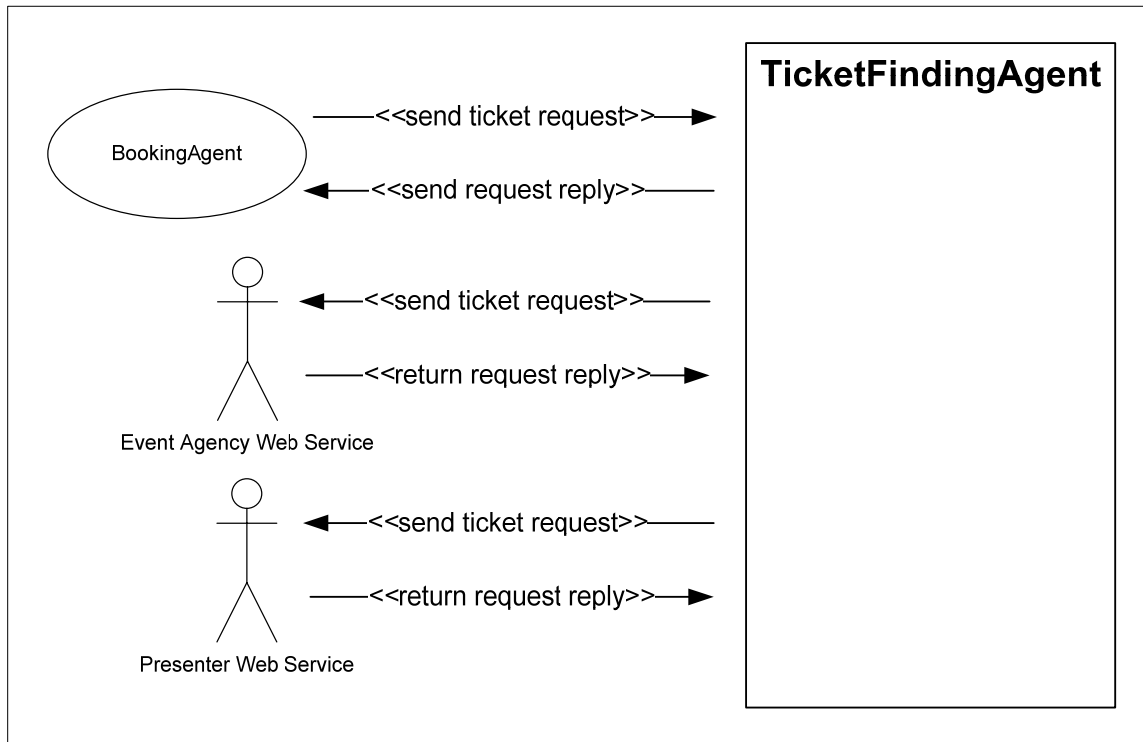


Figure 12: Use case TicketFindingAgent.

A Brief Description	BookingAgent forwards ticket request to TicketFindingAgent who as to check with external web services.	
B Precondition	Event information must be valid and not be found in the database of the system.	
C Postcondition	-	
D Processing Step		
1	TicketFindingAgent gets a ticket request from the BookingAgent for tickets that can not be found in the system's database.	
2	TicketFindingAgent receives request.	
3	TicketFindingAgent sends the ticket request to external web service of the event agency.	
4	Event agency web service sends reply as answer to the ticket request to the TicketFindingAgent.	
5	TicketFindingAgent sends the ticket request to external web service of event presenter.	
6	Presenter web service sends reply as answer to the ticket request to the TicketFindingAgent.	
7	TicketFindingAgent sends overall reply about the ticket request to the BookingAgent.	
E Exceptions		
3a	Connection to web service of event	Error message is generated saying that external

	agency is not possible.	system is not accessible. Error code is sent to BookingAgent. The use case is terminated.
5a	Connection to web service of presenter is not possible	Error message is generated saying that external system is not accessible. Error code is sent to BookingAgent. The use case is terminated.
F Relationships		
Initiating	BookingAgent	
Collaborating	BookingAgent, Presenter Web Service, Event Agency Web Service	
G Data Requirements		
Data required for TicketFindingAgent	(Event name OR event category OR event location OR event date) OR number of tickets from the BookingAgent.	

Table 2: Use case description TicketFindingAgent.

PaymentAgent.

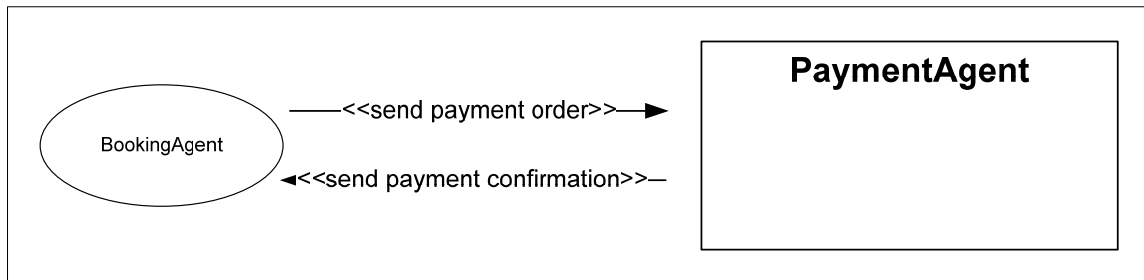


Figure 13: Use case PaymentAgent.

A Brief Description	BookingAgent forwards ticketing information to PaymentAgent to handle payment process for ticket purchase.	
B Precondition	Customer must confirm his intention to purchase tickets.	
C Postcondition	Payment must be confirmed by the system.	
D Processing Step		
1	PaymentAgent gets payment request from the BookingAgent for tickets the customer has selected.	
2	PaymentAgent receives request.	
4	PaymentAgent processes payment by using payment information either from the profile database or entered by the user.	
3	PaymentAgent sends payment confirmation to BookingAgent.	
E Exceptions		
1a	Connection is not possible.	Error message is generated saying that agent is not accessible. Error code is sent to BookingAgent. The use case is terminated.

F Relationships	
Initiating	BookingAgent.
Collaborating	BookingAgent
G Data Requirements	
Data required for PaymentAgent	Total price and customer information (and transaction ID).

Table 3: Use case description PaymentAgent.

TicketManagementAgent.

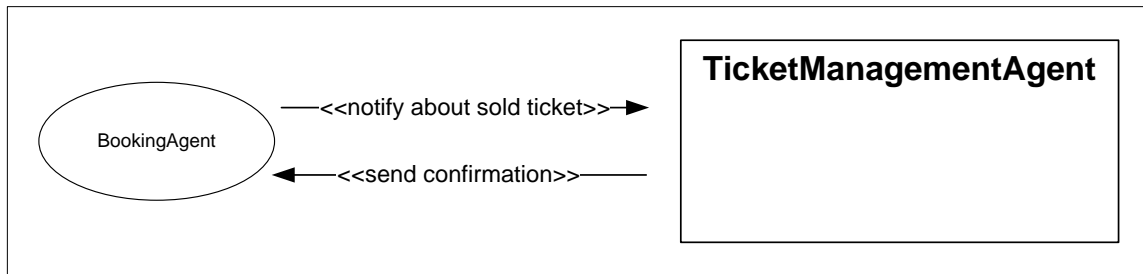


Figure 14: Use case TicketManagementAgent.

A Brief Description	BookingAgent sends notification about sold tickets to TicketManagementAgent to handle sold tickets.	
B Precondition	Payment process must be successful.	
C Postcondition	-	
D Processing Step		
1	BookingAgent sends notification about sold tickets to TicketManagementAgent.	
2	TicketManagementAgent removes tickets from data repositories and initializes shipment of tickets.	
3	TicketNotificationAgent sends notification to customer.	
4	TicketManagementAgent sends confirmation about successful handling to BookingAgent.	
E Exceptions		
1a	Connection is not possible.	Error message is generated saying that agent is not accessible. Error code is sent to BookingAgent. The use case is terminated.
F Relationships		
Initiating	BookingAgent.	
Collaborating	BookingAgent	
G Data Requirements		

Data required for TicketManagementAgent	Number and event for tickets.
--	-------------------------------

Table 4: Use case description TicketManagementAgent.

PDatabaseAgent.



Figure 15: Use case PDatabaseAgent.

A Brief Description	PDatabaseAgent handles communication with the profile database.	
B Precondition	Customer ID must be known.	
C Postcondition	-	
D Processing Step		
1	BookingAgent sends profile request for a specific customer to PDatabaseAgent.	
2	PDatabaseAgent processes request and handles database access.	
3	PDatabaseAgent sends database query to the profile database.	
4	The profile database returns a result for the query.	
5	PDatabaseAgent returns the result of the query to the BookingAgent, either profile information or a message that no profile exists for the customer (standard <i>no_success</i> message).	
E Exceptions		
1a	Connection is not possible.	Error message is generated saying that agent is not accessible. Error code is sent to BookingAgent. The use case is terminated.
3a	Database connection is not possible.	Error message is generated saying that profile database is not accessible. Error code is sent to BookingAgent. The use case is terminated.
F Relationships		
Initiating	BookingAgent.	
Collaborating	BookingAgent, Profile-Database.	
G Data Requirements		
Data required for PDatabaseAgent	Customer ID.	

Table 5: Use case description PDatabaseAgent.

EDatabaseAgent.

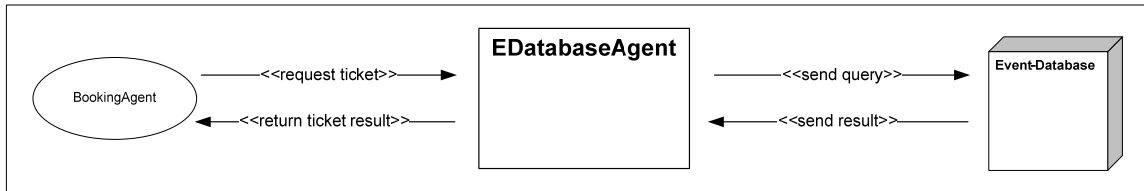


Figure 16: Use case EDatabaseAgent.

A Brief Description		EDatabaseAgent handles communication with the event database.
B Precondition		Event information must be known.
C Postcondition		-
D Processing Step		
1	BookingAgent sends event request for a specific event to EDatabaseAgent.	
2	EDatabaseAgent processes request and handles database access.	
3	EDatabaseAgent sends database query to the event database.	
4	The event database returns a result for the query.	
5	EDatabaseAgent returns the result of the query to the BookingAgent, either the desired ticket information or a message that no events for the desired criteria have been found.	
E Exceptions		
1a	Connection is not possible.	Error message is generated saying that agent is not accessible. Error code is sent to BookingAgent. The use case is terminated.
3a	Database connection is not possible.	Error message is generated saying that event database is not accessible. Error code is sent to BookingAgent. The use case is terminated.
F Relationships		
Initiating	BookingAgent.	
Collaborating	BookingAgent, Event-Database.	
G Data Requirements		
Data required for EDatabaseAgent	(Event name OR event category OR event location OR event date) OR number of tickets from the BookingAgent.	

Table 6: Use case description EDatabaseAgent.

5.2 Inter-Agent Communication Specification

In the proposed system, the communication architecture should be able to handle the communication between different agents. SOAP (Simple Object Access Protocol) provides the simplest communication system to fulfill this requirement because it enables

different agents with different internal structures to share a common communication language. It allows the communication between agents and agents and between agents and web services.

In detail, SOAP provides a simple and lightweight mechanism for exchanging structured and typed information between peers in a decentralized, distributed environment. Doing this it does not itself define any application semantics such as a programming model or implementation specific semantics; rather it defines a simple mechanism for expressing application semantics by providing a modular packaging model and encoding mechanisms for encoding data within modules. The SOA Protocol supports XML document exchange, Remote Procedure Call (RPC) and specifies a wire protocol for facilitating highly distributed applications. SOAP is similar to DCOM and CORBA in that it provides an RPC mechanism for invoking methods remotely. The difference is that it is a protocol based on open XML standards and XML document exchange rather than being an object model relying on proprietary binary formats. Both DCOM and CORBA use binary formats for their payload. Moreover, the use of CORBA would put an unnecessary extra pressure on the system because of its complexity [6]. The SOAP gateway performs a similar function to DCOM and CORBA stubs – translating messages between the SOAP protocol and the language of choice. As a result, SOAP offers vendor, platform, and language independence. With SOAP, developers can easily bridge applications written with COM, CORBA, or Enterprise JavaBeans.

One benefit for us in this is that it works well with network firewall what is the major advantage over other distributed protocols like GIOP/IIOP or DCOM (mentioned before) which are normally filtered by firewalls. Firewalls are a very important aspect in networks for security reasons so this was an important point for us. In our OTS the communication encodes a HTTP header (as today's internet standard) and an XML message (widespread acceptance and open source (cost free) development efforts). This allows us to communicate with external web services [7].

Basically a SOAP message is contained in a so called *envelope* (framework that describes what is in a message and how to process it). This envelope contains two additional sections: the *header* (contains relevant information about the message, e.g. date, or authentication information) and the *body* (contains the message) of the message. Additionally there are a set of encoding rules and conventions for representing remote procedure calls and responses [8].

Every agent must have an agent name and an agent locator which includes the transport descriptions for the agent. The transport description represents the standard of transportation, such as SMTP, IIOP or HTTP.

One more practical aspect for our decision is that SOAP is supported by a large number of freely available tools.

In the following the input and output messages for the system agents are described based on XML specifications.

TicketRequest.

Request
<pre><EventTicketRequest> <Event>string</Event> <Date>date</Date> <Location>string</Location> <NumberOfTicketsRequested>integer</NumberOfTicketsAvailable> </EventTicketRequest></pre>
Response
<pre><EventTicketResponse> <TicketID>Integer</TicketID> <Event>string</Event> <Date>date</Date> <Location>string</Location> <NumberOfTicketsAvailable>integer</NumberOfTicketsAvailable> <Prize>Float</Prize> </EventTicketResponse></pre>

ProfileRequest.

Request
<pre><ProfileRequest> <CustomerID>Integer</CustomerID> </ProfileRequest></pre>
Response
<pre><ProfileResponse> <CustomerName>String</CustomerName> <CustomerAddress>String</CustomerAddress> <AccountInformation>String</AccountInformation> <CreditCardInformation>String</CreditCardInformation> </ProfileResponse></pre>

PaymentRequest.

Request
<pre><PaymentRequest> <TransactionID>Integer</TransactionID> <CustomerID>Integer</CustomerID> <Amount>Float</Amount> </PaymentRequest></pre>
Response

TicketManagementRequest.

Request
<pre><TicketManagementRequest> <TicketID>Integer</TicketID> <NumberOfTickets>Integer</NumberOfTickets> <Event>string</Event> <Date>date</Date></pre>

<pre> <Location>string</Location> <TotalPrize>Float</TotalPrize> </TicketManagementRequest> </pre>
Response

5.3 Web Service Communication

The communication to the external web services has to follow a standard WSDL definition. The following listing shows the proposed XML message format for the communication between OTS and external web services (generic scheme: <EVENT_PROVIDER_URL> has to be substituted by URL of the external event provider where web services are hosted):

```

<?xml version="1.0"?>
<definitions name="EventWebservice"

targetNamespace="http:// <EVENT_PROVIDER_URL>//eventwebservice.wsdl"
  xmlns:tns="http:// <EVENT_PROVIDER_URL>//eventwebservice.wsdl"
  xmlns:xsd1="http:// <EVENT_PROVIDER_URL>//eventwebservice.xsd"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">

  <types>
    <schema targetNamespace="http://<EVENT_PROVIDER_URL>//eventwebservice.xsd"
      xmlns="http://www.w3.org/2000/10/XMLSchema">
      <element name="EventRequest">
        <complexType>
          <all>
            <element name="event_name" type="string"/>
            <element name="date" type="string"/>
            <element name="location" type="string"/>
            <element name="number_of_tickets_requested" type="integer"/>
          </all>
        </complexType>
      </element>
      <element name="EventReply">
        <complexType>
          <all>
            <element name="ticket_id" type="string"/>
            <element name="event_name" type="string"/>
            <element name="date" type="string"/>
            <element name="location" type="string"/>
            <element name="number_of_tickets_available" type="integer"/>
            <element name="number_of_tickets_requested" type="integer"/>
            <element name="prize" type="float"/>
          </all>
        </complexType>
      </element>
    </schema>
  </types>

  <message name="GetEventInput">
    <part name="body" element="xsd1:EventRequest"/>
  </message>

  <message name="GetEventOutput">
    <part name="body" element="xsd1:EventReply"/>
  </message>

  <portType name="EventWebservicePortType">
    <operation name="GetEvent">
      <input message="tns:GetEventInput"/>

```

```

        <output message="tns:GetEventOutput" />
    </operation>
</portType>

<binding name="EventWebservice SoapBinding" type="tns: EventWebservice PortType">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="GetEvent">
        <soap:operation soapAction="http://<EVENT_PROVIDER_URL>//GetEvent" />
        <input>
            <soap:body use="literal" />
        </input>
        <output>
            <soap:body use="literal" />
        </output>
    </operation>
</binding>

<service name=" EventWebservice">
    <documentation>External Event web service</documentation>
    <port name=" EventWebservice Port" binding="tns:EventWebservice Binding">
        <soap:address location="http://<EVENT_PROVIDER_URL>//eventwebservice"/>
    </port>
</service>
</definitions>

```

5.4 Data Specification

The basic two data structures of the system are the event (stored in the event database or handled by external web services) and the customer (stored in the profile database).

Event database.

Field	Description	Type
EventID	Unique identifier of this event.	Integer
Name	Name of this event.	String
Type	Type of event, e.g. pop, rock, opera.	String, enum.
Date	Date of event.	Date
Time	Time of event.	Time
Location	Location of event.	String
Presenter	Presenter of event.	String
Prize	Prize for one ticket of this event.	Float

Table 7: Data description of event database.

Profile database.

Field	Description	Type
ProfileID	Unique identifier of this profile.	Integer

First_Name	First name of customer.	String
Last_Name	Last name of customer.	String
Street	Street of customer.	String
ZIP	ZIP of customer.	String
Town	Town of customer.	String
Province	Province of customer.	String
Creditcard_Number	Credit card number of customer.	String
Expiry_Date	Expiry date of customer's credit card.	String
Creditcard_Holder	Name of credit card holder.	String
Bank	Bank of customer.	String
Account_Number	Account number of customer.	String
Preferred_Events	Preferred events of customer.	String, enum.

Table 8: Data description of profile database.

6 Conclusion

This document described the analysis and design of an online ticket system by using the concept of agent-based software development and engineering. It presented the system design, a technological overview and a detailed specification.

The proposed multi-agent based network ticket system enhances current ticket systems used to sell tickets. Such a system is not limited to only the event ticket domain but can be extended to other areas as well. The primary difference between the current event ticket systems available for customers today and the proposed system is the use of a multi-agent system (because of the benefits mentioned in chapter 2) that is responsible for the handling of the complete ticket process, from ticket request to payment. Another difference is the access to world wide events. The automatic agent-based ticket system is an important step towards intelligent, more customer friendly event ticket selling.

References

- [1] F. Giunchiglia, J. Mylopoulos, and A. Perini, "The Tropos Software Development Methodology: Processes, Models and Diagrams," presented at First International Joint Conference on Autonomous Agents and Multiagent Systems, Bologna, Italy, 2002.
- [2] E. Cerami, *Web services essentials*. Cambridge, MA: O'Reilly & Associates, 2002.
- [3] J. Newmarch, *A Programmer's Guide to Jini Technology*. Berkley, CA: Apress, 2000.
- [4] W3C, "Web Service Definition Language (WSDL)." www.w3c.org/TR/wsdl/, 2005.
- [5] T. Kyte, *Expert Oracle Database Architecture: 9i and 10g Programming Techniques and Solution*. Berkley, CA: Apress, 2005.
- [6] S. Ahmed, *CORBA Programming Unleashed*. Indianapolis, IA: Sams Publishing, 1999.
- [7] S. Graham, *Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI*. Indianapolis, IA: Sams Publishing, 2004.
- [8] W3C, "SOAP Specifications." <http://www.w3.org/TR/soap/>, 2005.