



Movie Ticket Dispenser Website (web-app)

MTD

SENG 697: Agent-Based Software Engineering

by

Mehrnoush Alemzadeh

Instructor:

Dr. B. H. Far

Department of Computer Science and Electrical
Engineering

University of Calgary
Calgary, Alberta T2N 1N4
Canada

October 2006

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	10
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 Implementation Overview	20
4.1 Database and Programming Language	20
5 Detailed Design	21
5.1 Use Cases	21
5.2 Inter-Agent Communication Specification	27
5.3 Data Specification	29
6 Conclusion	30
References	31

List of Figures

Figure 1: Informal structure of the MTD.....	8
Figure 2: <i>Tropos</i> actor diagram for the Movie Ticket Dispenser.	12
Figure 3: Refined <i>Tropos</i> actor diagram for the actor <i>MTD</i>	13
Figure 4: Architectural structure of the MTD.....	14
Figure 5: Internal structure of a BookingAgent.....	16
Figure 6: Internal structure of DatabaseAgent.....	17
Figure 7: Internal structure of TicketFindingAgent.....	18
Figure 8: Internal structure of PaymentAgent.	19
Figure 9: Use case BookingAgent.	21
Figure 10: Use case TicketFindingAgent.	23
Figure 11: Use case PaymentAgent.	25
Figure 12: Use case DatabaseAgent.	26

List of Tables

Table 1: Use case description for BookingAgent.....	22
Table 2: Use case description of TicketFindingAgent.....	24
Table 3: Use case description PaymentAgent.....	25
Table 4: Use case description DatabaseAgent.....	26
Table 5: Data description of movie.....	29
Table 6: Data description of user profile.	29

Abstract

In this report we will describe a web-based multi-agent Movie Ticket Dispenser (MTD). Through which the customer can buy movie tickets on web. The system has access to movies in an internal data base as well as the new releases made available by providers using web services. Several agents are introduced to support these tasks. This report will present the business case, the architectural description, a technological overview, and design details.

1 Introduction

The entertainment industry is one of the most profitable sectors in the business world. People always spent money for being entertained, and so will they in the future. The only problem is how people get to know about all the currently presented movies as well as the future ones, and then, in case they know, how they can get tickets. This problem used to be solved by theaters' ticket kiosks at the Showtime or by phone reservation which are in most cases restricted by time, location, and – most important – by availability of tickets. Because of that reason the following document will describe the implementation details of a Movie Ticket Dispenser (MTD). It is intended to work as a web-based multi-agent application. The system will allow the user to buy tickets online for movies that are either stored in the system's own database or available using web services offered by the Entertainment companies, but accessed by the MTD. Bookings can be made independent of time and location.

The report will be structured 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. An implementation overview follows in the fourth chapter and the design details of the system are introduced in chapter 5. Finally we will draw a conclusion at the end of this paper.

The following terminology will be used throughout the document:

<i>Customer</i>	Person that uses the application to purchase tickets.
<i>Admin</i>	Entity outside the system that has access to the system.
<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 many other industries and businesses, customers in entertainment industry ask for services that are more and more available independent from time and location. People want to buy tickets 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 movie industry.

Customers are using the internet as a medium to purchase many things for which they used to go to real stores, and so do they for tickets. When searching for movies customers need a ticket dealer to show and sell them what is available according to their wishes, or they consult specific websites of theatres, etc. Here, intelligent agents have a great potential in helping the customer to get what he wants, whenever he wants.

We use agent-based Software engineering approach to build MTD as a multi-agent system. Agent-based systems provide expandability and scalability as well as support for heterogeneous mobile structures. Additionally, our MTD 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:

- collaborative work, roles, responsibilities, and goal directness
- Reactive behavior of components in MTD.
- autonomous single agents in MTD work together to achieve root goal (see *Tropos* models Figure 2 and Figure 3)
- Entities in MTD can communicate and interact with other entities and artifacts in the outer environment (like web services).
- MTD agents will be knowledgeable and able to manage the information on their own.
- MTD 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 Movie Ticket Dispenser is a multi-agent web-based system designed to purchase tickets for movies according to the preference of the customer and independent of time and location of the customer. Using the system must be as simple as possible so we propose a simple web interface that can run on any web browser (on a PC, WAP enabled cell phones, etc.). No download of special client side agents is necessary. The system does not only offer movies that are stored in the system's internal database but also the ones provided by other providers (either other booking agencies or movie studios) via their own web services.

What is sold by our system is one or more ticket(s) for a certain (or more) movie(s). The application is able to deal with different booking agencies, sponsors, 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 database of the system can either be made by the operator of the MTD or by external entertainment agencies.

The following figure shows a general description of the system.

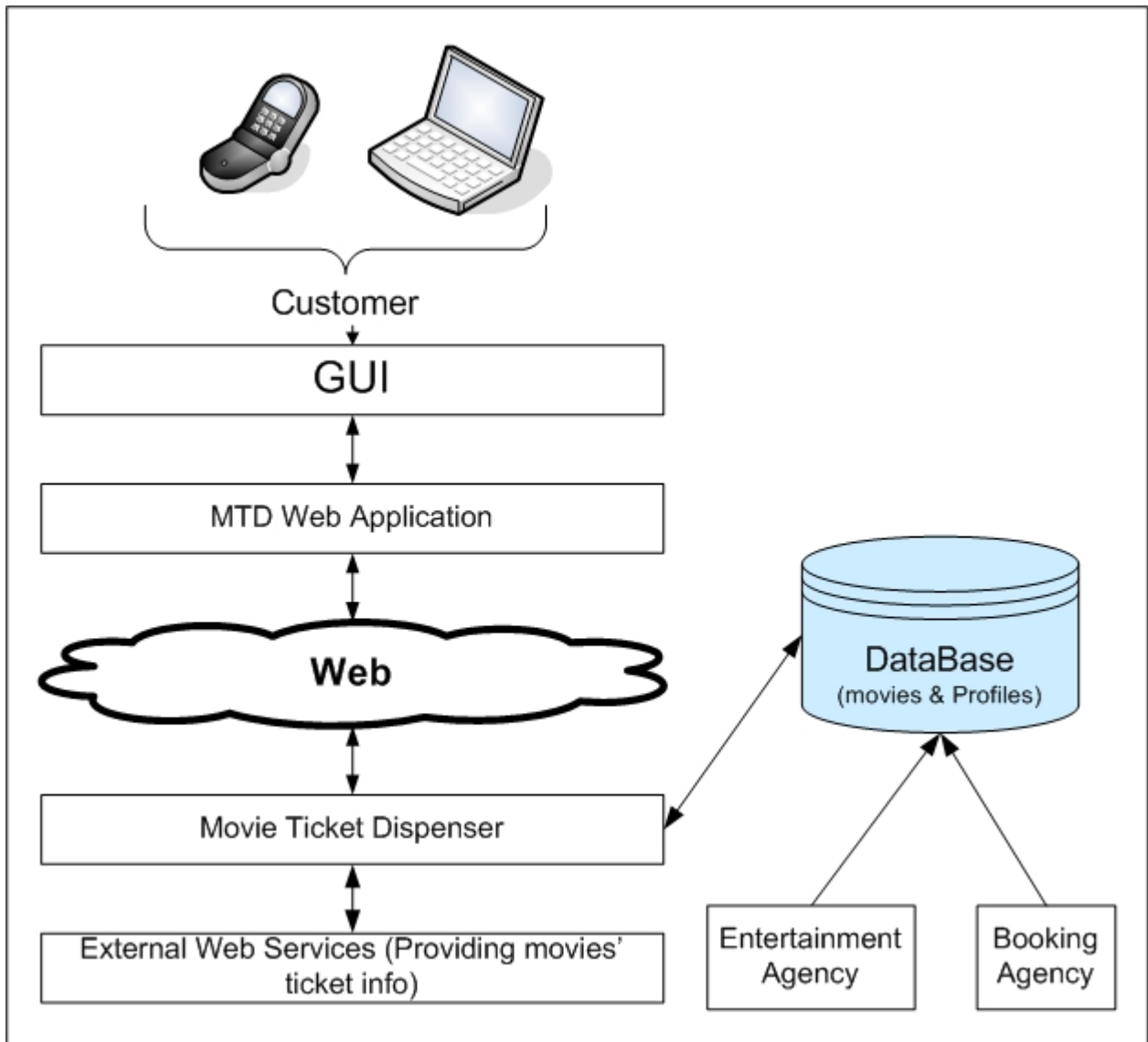


Figure 1: Informal structure of the MTD.

2.3 Assumptions

For the development of the MTD software system we will consider the following assumptions:

The system maintains a user profile (*User-Database* in Figure 1) for each user but not necessarily. A user profile can be helpful for frequent customers to store customer related information (e.g. payment information, preferences for customized offers and mail advertising, etc.). Preferences could be any type of movies, locations, or days.

The custom asks for movies using a graphical user interface that can be used with every web browser available on every device with internet access (PC, WAP enables cell phone, etc.). The interface captures the information and the system retrieves movies based on that information. For frequent users, profile information is also taken into consideration.

The customer can buy as many tickets as he wants and available for one movie. He also can buy tickets for different movies at one time. The customer should be able to buy tickets for movies from literally anywhere.

The system may have access to existing infrastructures of other booking agencies, and entertainment agencies (movie producers or sponsors). 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 system must fulfill the following requirements in order to meet the ultimate goal of the system which is the customer satisfaction by providing a broad ticket service independent of time and location:

The MTD is required to:

- R1. Provide movie information and schedules as well as searching features.
- R2. Deal with different movie producers, booking agencies, local theaters and movie presenters.
- R3. Be able to book the desired ticket upon the approval by the customer.
- R4. Be able to send a booking and payment confirmation to the customer after approval of booking and payment.
- R5. Allow the customer to create a user profile if desired by the customer.
- R6. Be able to use customer information to process payment, when a user profile exists.
- R7. Allow entries to the system's database by the operator of MTD and by external providers.
- R8. Make the Tickets available at any location.

2.5 Wish List of Not Implemented Features (Future Requirements)

There are some features not implemented in this version of MTD but planned to be added in future versions (to achieve a higher level of customer satisfaction):

The MTD should:

- W1. Be able to send a reminder of the booked tickets to the customer prior of the time, When a user profile exists,.
- W2. Notify the customer automatically, if a ticket is canceled.
- W3. Once a seat is selected and during the booking process the seat must be locked in Database to prevent overbooking of one seat by multiple users in the system.
- W4. Be able to send customized offers of upcoming releases to the customer, when a user profile exists.
- W5. Guarantee a high level of security for payment.
- W6. Be able to locate existing web services, and their selection should be based on trusted rating services.

3 Top Level System Design

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

- The 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 root goals.
- Analyzes root goals from the actor's perspective.
- Generated sub goals are delegated to other actors (which may require the introduction of new actors).
- Analysis is carried out concurrently with respect to each root goal.
- Ends when all goals have been dealt with to the satisfaction of the respective actors.

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

3.1 Roles and Actors

In this project we use *Tropos* approach to define the actors and top level goals of the system. Figure 2 shows the *Tropos* actor diagram based on the mentioned actors and root goals, 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 symbolize hardgoals, and circles describe actors and roles.

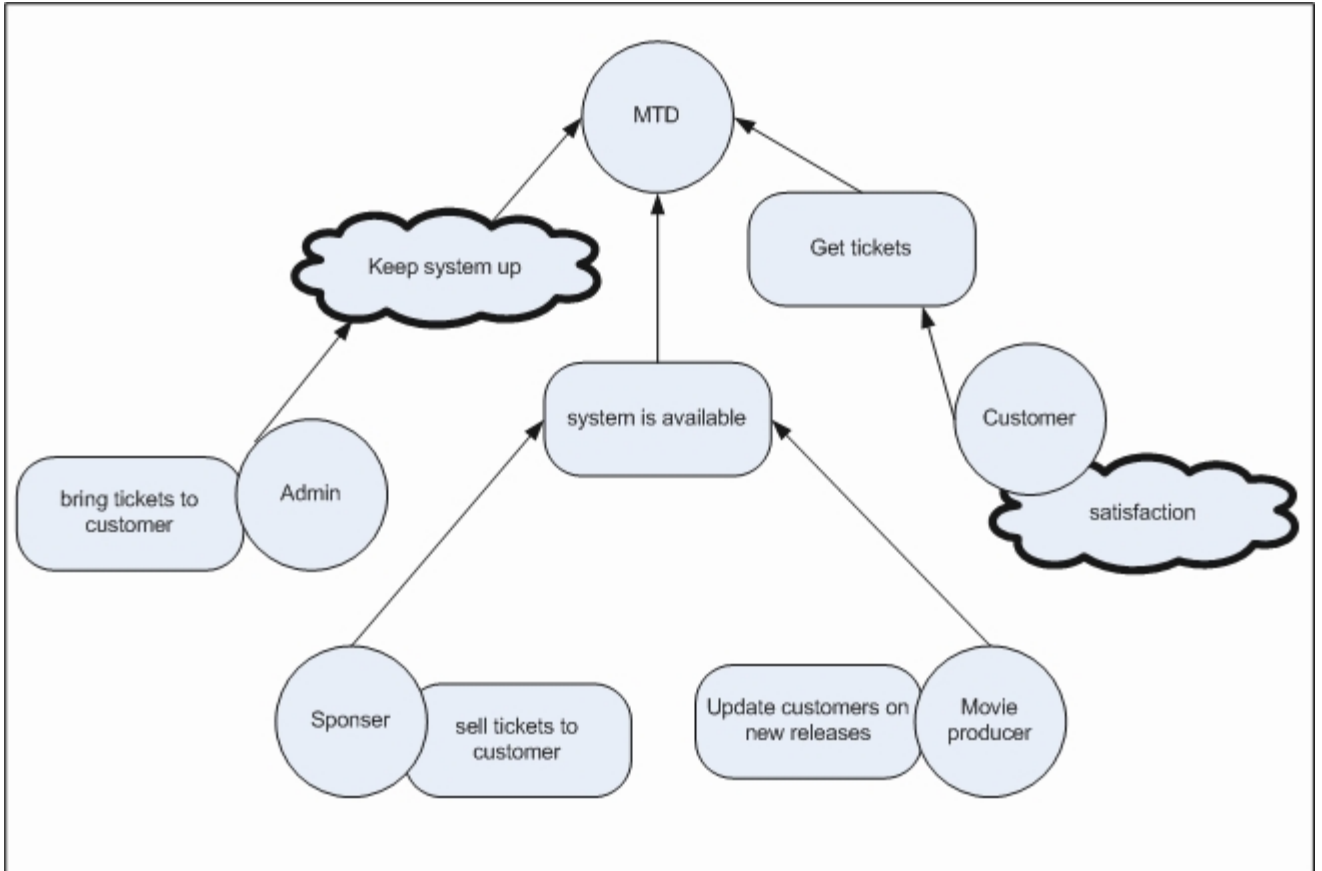


Figure 2: Tropos actor diagram for the Movie Ticket Dispenser.

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

- **Customer.** Who selects a movie (or more), buys and pays for the ticket. And may create a profile in the system.
- **Admin.** Who can enter movie information into the database. Who can be an external movie producer or the operator of the MTD itself.
- **Movie sponsor.** A company that provides a movie that can be purchased using the MTD. Movie providers can be either other agencies that have contracts with producers or the theaters directly.
- **MTD.** The system itself will be the main actor to fulfill all the requirements for the actors.

With more detailed analysis we refined the actor *MTD* (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.

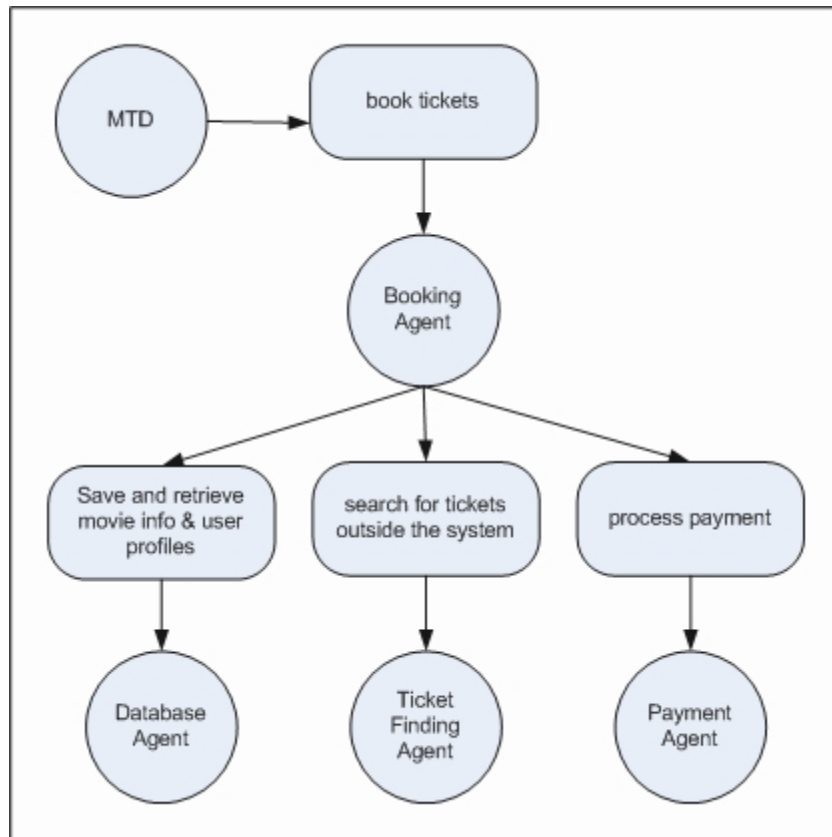


Figure 3: Refined *Tropos* actor diagram for the actor *MTD*.

We can derive our agents (see Figure 4) by using this model.

3.2 Architecture Description

The architectural description of the MTD is depicted in Figure 4.

Users submit their requests to the system.

We have two different types of users: admins and customers:

- *System administrator*: enters movie info into the system Database by means of a web user interface.
- *Customer*: submits a request using the system GUI.

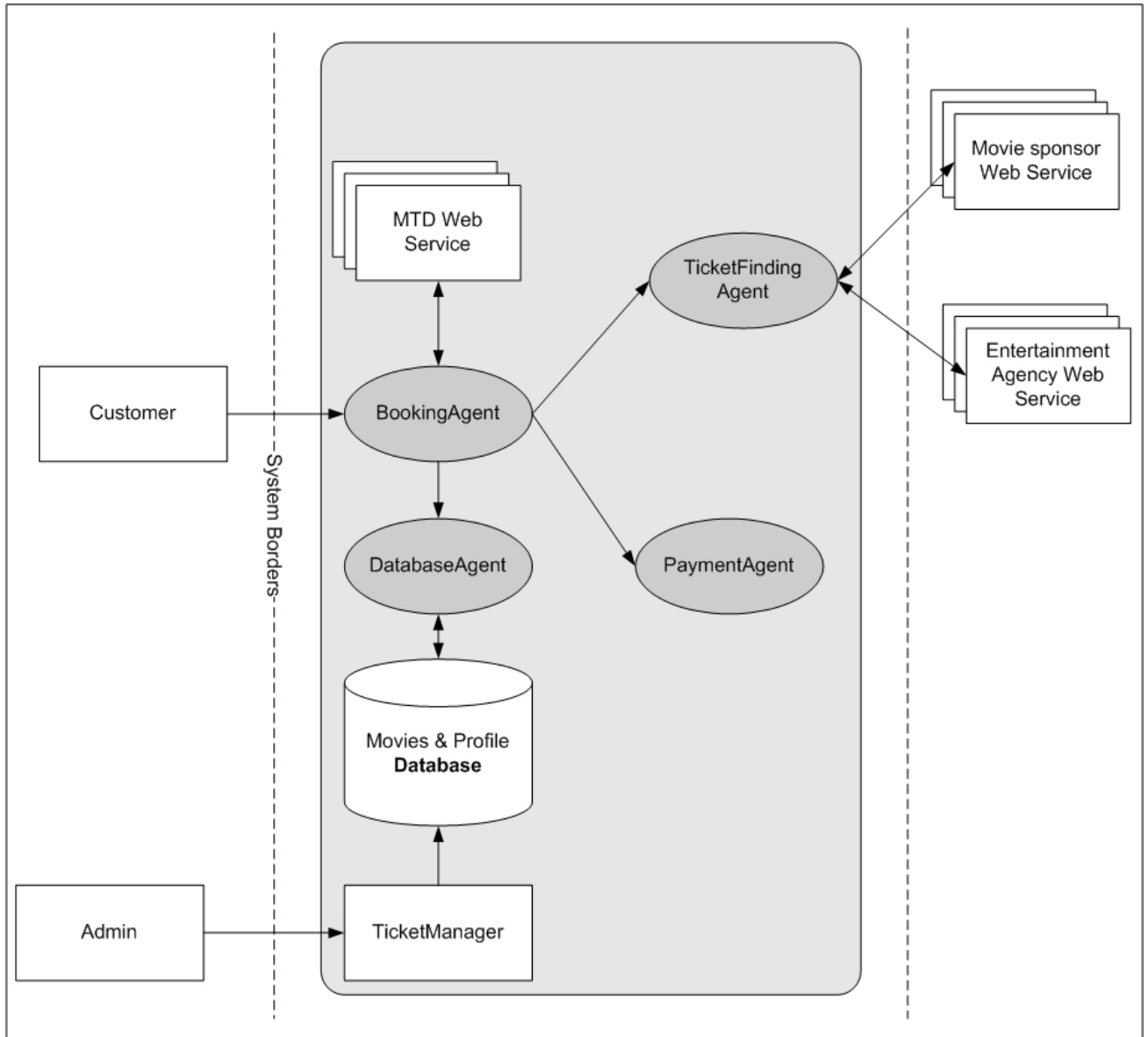


Figure 4: Architectural structure of the MTD.

Customer can brows the movie database as well as searching for a special movie according to several attributes like genre, rating, length of the movie, theaters, name of the director ,.....

He could also create a profile if he/she prefers, which enables use of additional features, which also makes the process of payment faster because of using the stored information. So it can be useful for regular users.

MTD as a multi-agent system has to interact with the web services from external providers if they own a web service. Otherwise this information should be entered into the system's database by the system admin.

3.3 Agent Description

MTD system has basically four agents, as visualized in Figure 3 and Figure 4, which will be described in further details in the following part:

BookingAgent. Which is responsible for receiving the customer requests through the web GUI. This agent manages the communication with other agents, and can be seen as a central point of delegation for the whole system. This agent requests info either from the internal database by sending the requests to the DatabaseAgent or from web services provided by external providers (by calling the TicketFindingAgent).

It is also responsible for displaying the results to the user, also represented in the web GUI. It sends requests to the agents to book selected items, and then creates a message that contains the information about the process and displays that as a confirmation to the user. Finally this agent has to trigger a search for the customer's profile to accelerate the payment process. If no profile is available, the customer must enter payment information by hand for each purchase. Figure 5 shows the internal architecture of the BookingAgent.

Component description:

Listener. Listens for incoming requests from the customer requesting for a movie, and works as an interface to the outside environment.

Interpreter. Has to parse and interpret the incoming XML messages (the standard transfer protocol used here is XML)

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

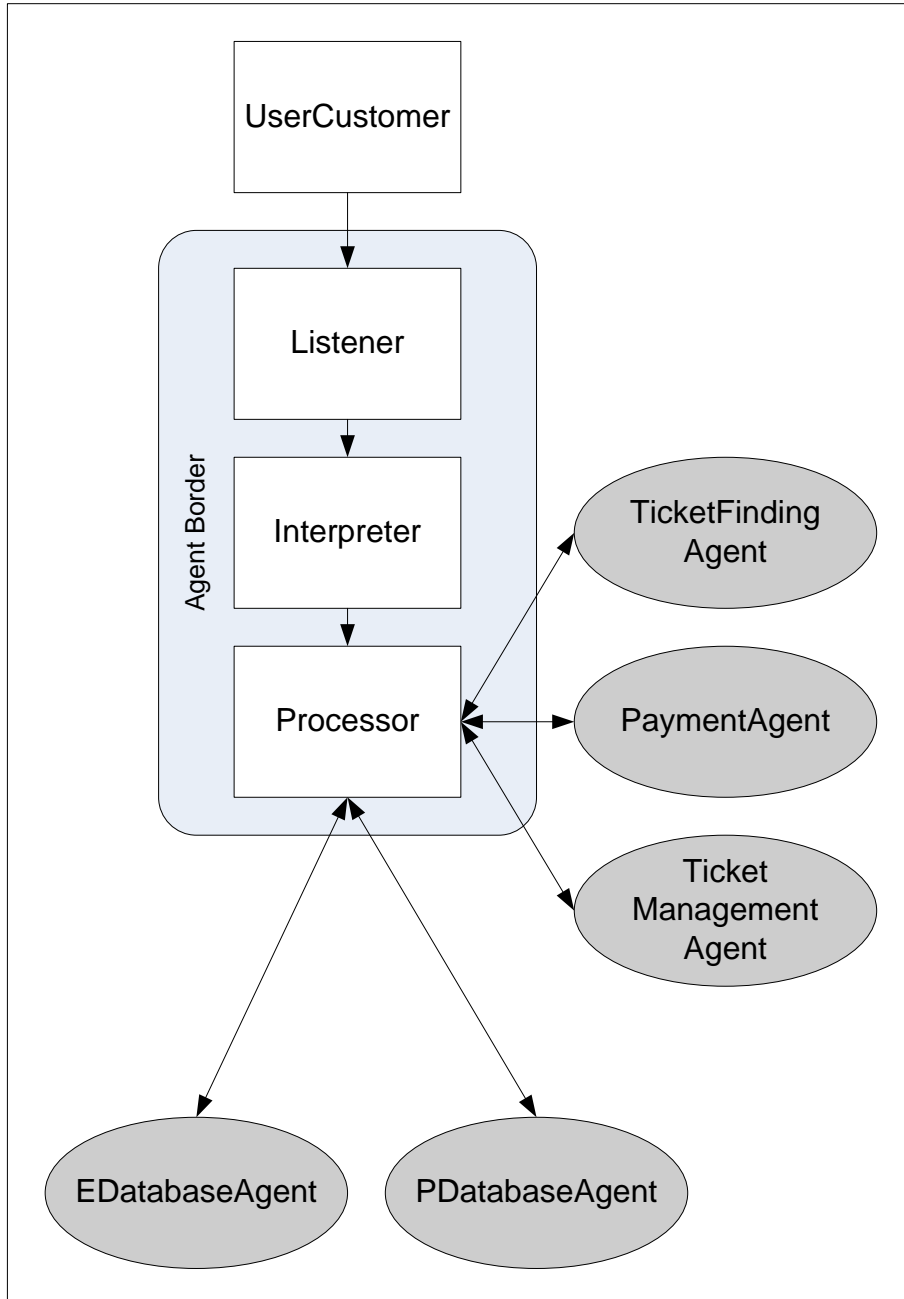


Figure 5: Internal structure of a BookingAgent.

DatabaseAgent. This agent (Figure 6) communicates with the BookingAgent. The DatabaseAgent is responsible for handling all enquiries that are related to the database. The Database can be either Movie or Profile Database. Requests are sent by the BookingAgent. The DatabaseAgent returns the item(s) found for each request (movie and number of tickets) if available. If no ticket(s) 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. The request could be also regarding customer profiles from the BookingAgent. The agent returns the data fields requested by the BookingAgent, and In case of no information available for a certain customer, a standard *no_success* notification.

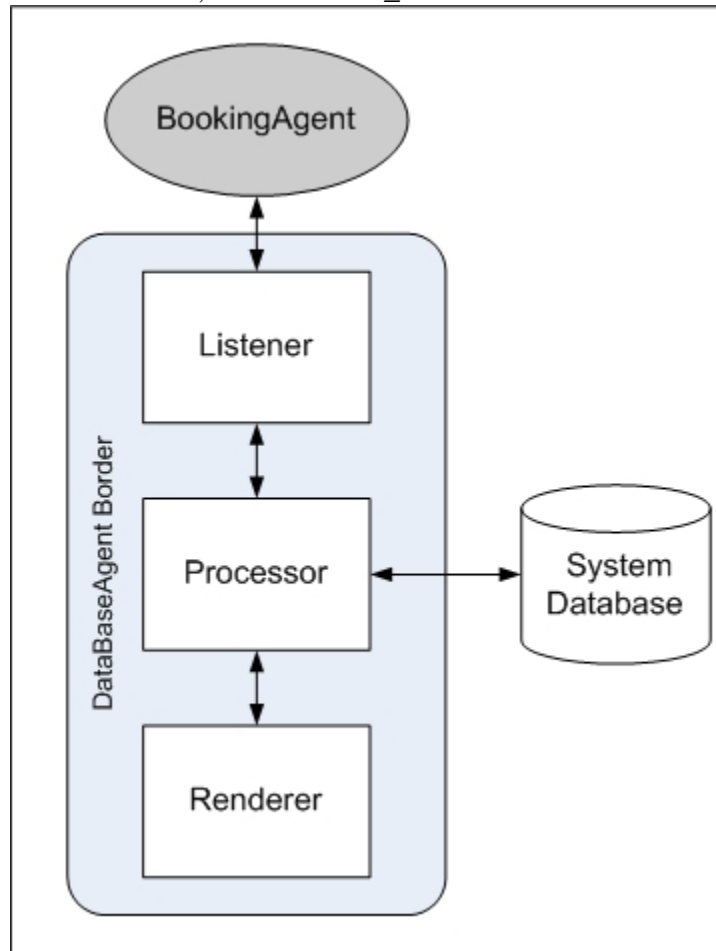


Figure 6: Internal structure of DatabaseAgent.

Component description:

Listener. Listens for any incoming requests from the BookingAgent.

Processor. Gets the parsed XML document and sends a request to the database (movie DB or Profile DB).

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

TicketFindingAgent. The TicketFindingAgent as shown in Figure 8 is responsible for finding appropriate tickets for the customer accessing external web services, if they exist. To fulfill this task it contacts the BookingAgent. The notification is sent by the BookingAgent when no tickets could be found in the system's own database. The requests are processed and sent 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.

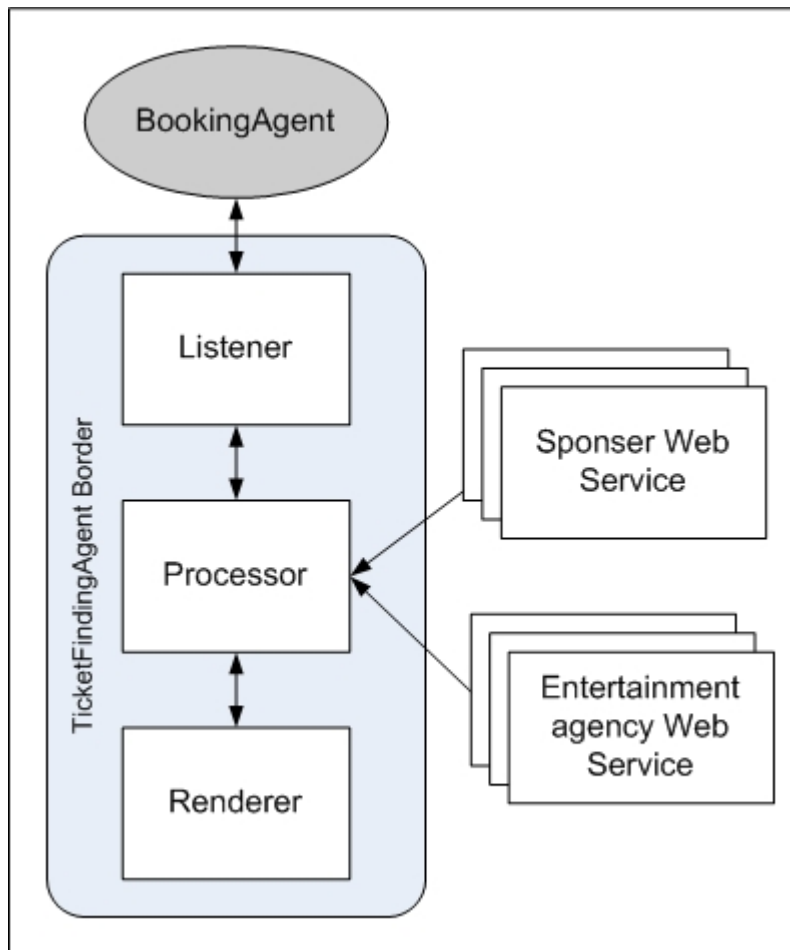


Figure 7: Internal structure of TicketFindingAgent.

Component description:

Listener. Listens for any incoming requests from the BookingAgent.

Processor. Gets the parsed XML document and calls the external web services.

Renderer. Renders the external data before sending them back to the calling BookingAgent as well as rendering the data received from the BookingAgent before sending them to the web services.

PaymentAgent. This agent, as shown in Figure 9, works in communication with the BookingAgent and is responsible for the payment process. Once a customer has selected and confirmed that she/he wants to purchase a ticket (or more), the PaymentAgent is notified by the BookingAgent.

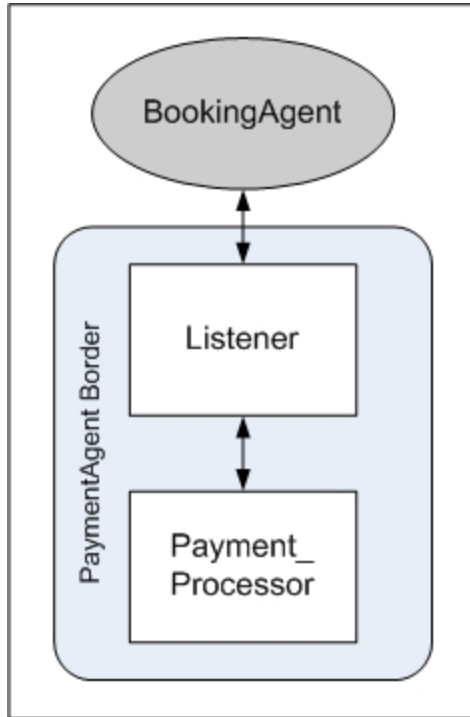


Figure 8: Internal structure of PaymentAgent.

Component description:

Listener. Listens for incoming payment requests from the BookingAgent.

Payment_Processor. Gets the parsed XML document and executes the payment functions for the current transaction.

4 Implementation Overview

We will give an overview of the technologies used to implement the MTD, as a web based system. This in turn makes us to consider web technologies and web services (for details see [2]) to build the system.

4.1 Database and Programming Language

The database will be a SQL server database and the programming language will be Java because of its compatibility with web technologies and the SOAP communication protocol [8]. The inter-agent communication will be XML-based. The web services must follow a common WSDL, also defined in XML (see the detailed design chapter).

JADE platform was chosen for the development because it provides features that support the product's:

- Compliance with FIPA for agent-based interoperability.
- Independent from underlying network and platform.
- Support for message content creation and management including XML.
- Support for Java, JSP, HTTP, IIOP.
- Graphical development environment supporting debugging, management and monitoring phase.
- Use of the *JINI Discovery (java code discovery approach)* to locate and interact with existing web services

5 Detailed Design

We will give a detailed description of the components of the MTD in the following chapter. Use cases for all the agents will be defined as well as the communication details and the data specifications.

5.1 Use Cases

We will present use cases for all agents in the following section.

BookingAgent.

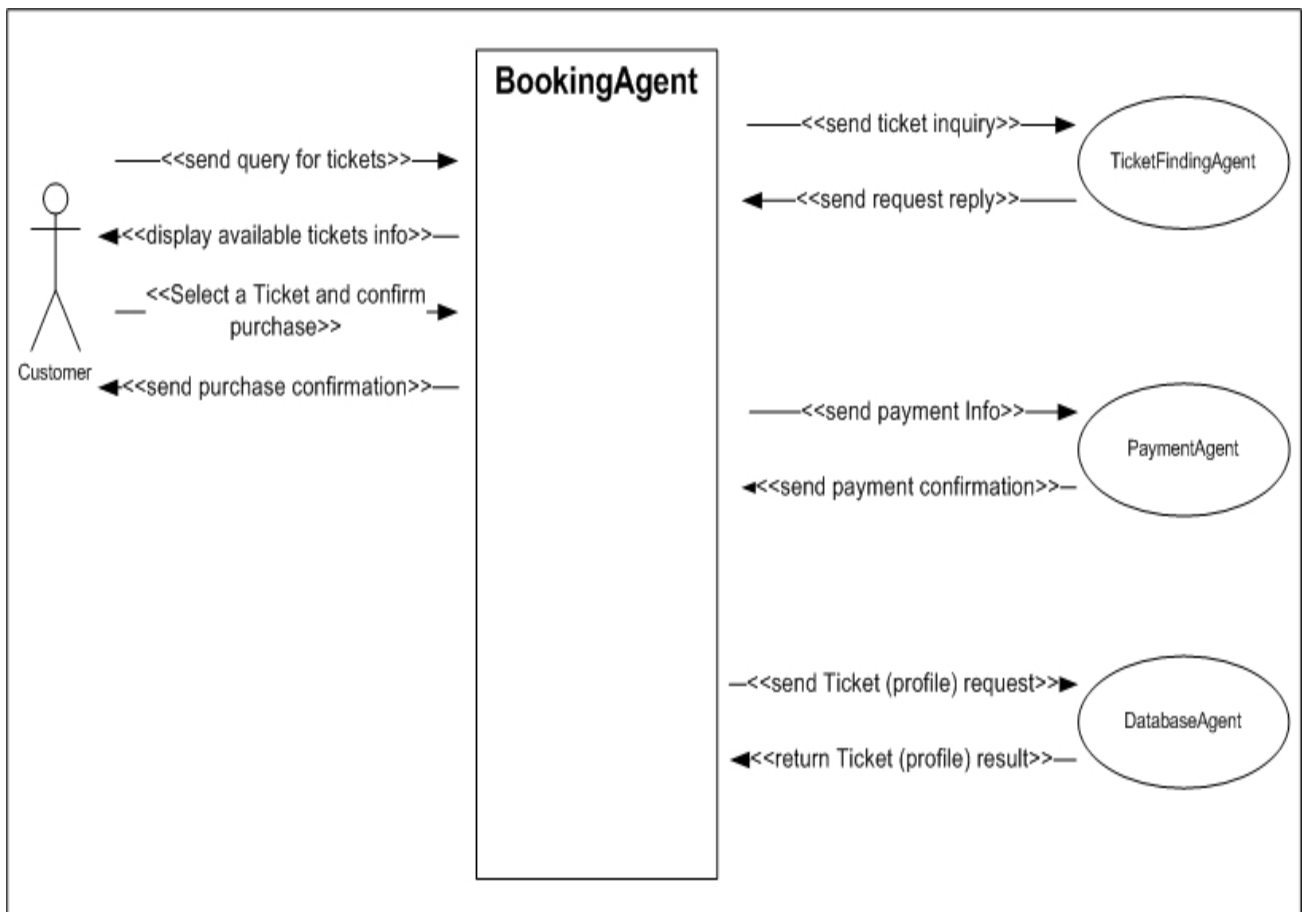


Figure 9: Use case BookingAgent.

Brief Description	Customer actor requests and purchases a ticket.	
Precondition	If there is any user profile, it is checked and loaded.	
Postcondition	If purchasing process was successful the ticket(s) will be booked out of the system.	
Processing Step		
1	Customer requests a certain ticket, on a certain date with a certain quantity.	
2	BookingAgent receives the request.	
3	BookingAgent checks if a profile exists for this customer by sending a query to the DatabaseAgent.	
4	DatabaseAgent returns profile information; or a <i>no_success</i> message if there is no profile	
5	BookingAgent sends ticket request to the TicketFindingAgent and the DatabaseAgent.	
6	BookingAgent receives reply from TicketFindingAgent and DatabaseAgent.	
7	BookingAgent displays the results to the customer.	
8	Customer selects tickets or cancels them based on the list shown in the browser.	
9	BookingAgent sends the ticket info to DatabaseAgent.	
10	BookingAgent notifies PaymentAgent to process payment.	
11	PaymentAgent returns payment confirmation.	
12	BookingAgent sends notification to DatabaseAgent to update customer information in case a profile exists.	
Exceptions		
1a	System is not accessible.	Error message is generated saying that system is not accessible. Use case is terminated.
7a	Customer cancels action.	Connection is closed, home page is shown.
Relationships		
Initiating	Customer.	
Collaborating	TicketFindingAgent, PaymentAgent, DatabaseAgent.	
Data Requirements		
Data required for BookingAgent	movie name OR category OR theater location OR ticket date OR number of tickets.	

Table 1: Use case decription for BookingAgent.

TicketFindingAgent.

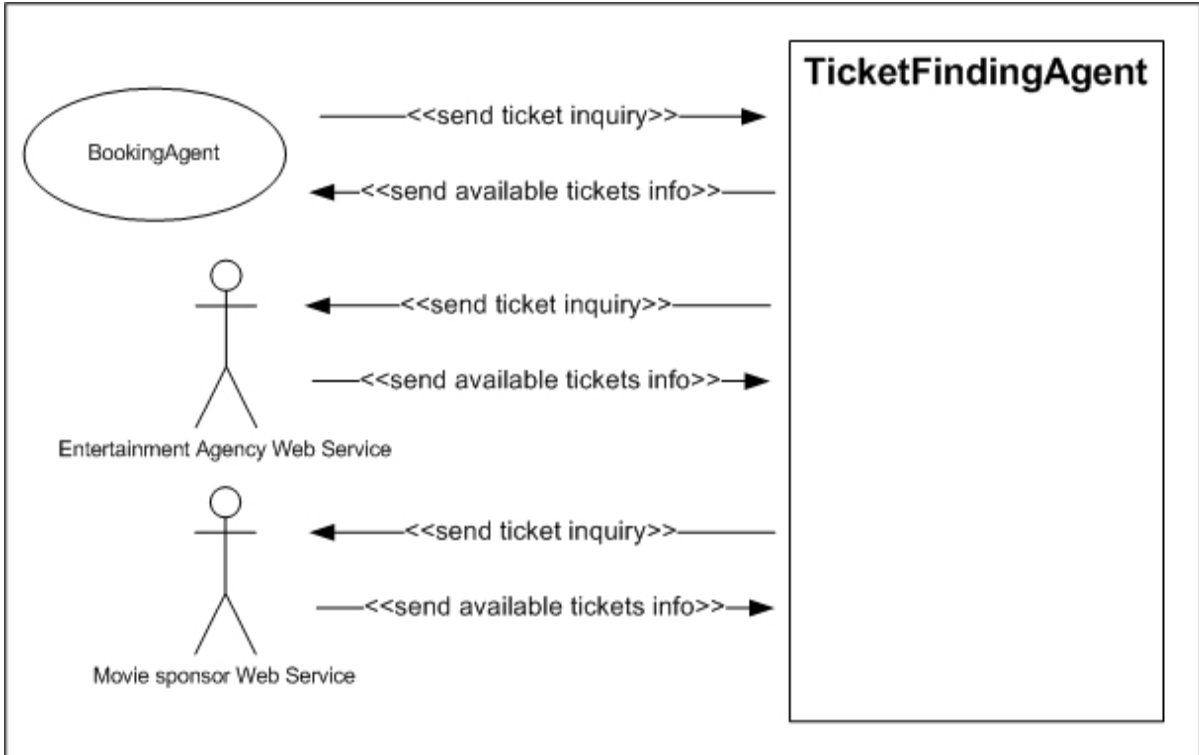


Figure 10: Use case TicketFindingAgent.

Brief Description	BookingAgent sends ticket request to TicketFindingAgent who checks with external web services.	
Precondition	Movie information must be valid and not be found in the database of the system.	
Postcondition	-	
Processing Step		
1	BookingAgent sends a ticket request to the TicketFindingAgent for tickets not currently in the system's database.	
2	TicketFindingAgent receives the request.	
3	TicketFindingAgent sends the ticket request to external web service of other agencies.	
4	BookingAgency web service sends reply as answer to the ticket request to the TicketFindingAgent.	
5	TicketFindingAgent sends the ticket request to external web service of the movie sponser.	
6	Sponser web service sends reply to the TicketFindingAgent.	
7	TicketFindingAgent sends the reply about the ticket request to the BookingAgent.	
Exceptions		
3a	Connection to web service of movie	Error message is displayed saying that external

	production agency cannot be established.	system is not accessible. Error code is sent to BookingAgent. Use case is terminated.
5a	Connection to web service of sponsor is not possible	Error message is displayed saying that external system is not accessible. Error code is sent to BookingAgent. Use case is terminated.
Relationships		
Initiating	BookingAgent	
Collaborating	BookingAgent, Sponsor Web Service, movie production Web Service	
Data Requirements		
Data required for TicketFindingAgent	(movie name OR category OR ticket date) OR number of tickets from the BookingAgent.	

Table 2: Use case description of TicketFindingAgent.

PaymentAgent.

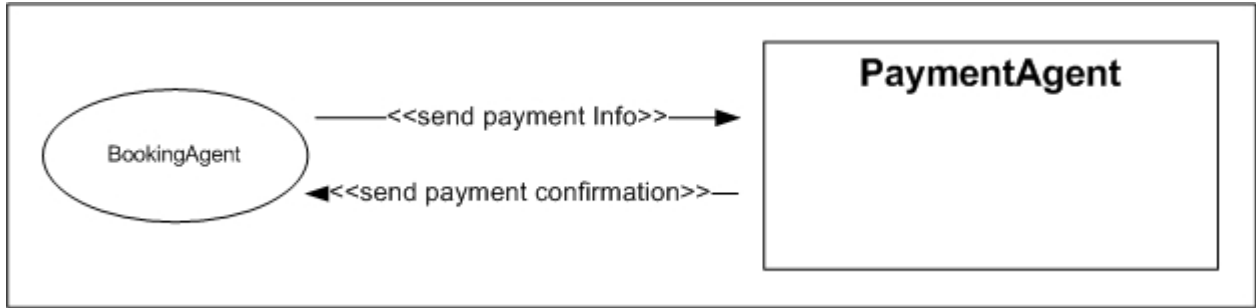


Figure 11: Use case PaymentAgent.

Brief Description	BookingAgent forwards ticketing information to PaymentAgent to handle payment process for ticket purchase.
Precondition	Customer confirms his purchased tickets.
Postcondition	BookingAgent confirms the payment
Processing Step	
1	BookingAgent sends a payment request for tickets the customer has selected to the PaymentAgent.
2	PaymentAgent receives request.
4	PaymentAgent processes payment by using payment information either from the user's profile from the database or entered by the user.
3	PaymentAgent sends payment confirmation to BookingAgent.
Exceptions	
1a	Connection could not be established. Error message is displayed saying that agent is not accessible. Error code is sent to BookingAgent. The use case is terminated.
Relationships	
Initiating	BookingAgent.
Collaborating	BookingAgent
Data Requirements	
Data required for PaymentAgent	Total price, customer information (and transaction ID).

Table 3: Use case description PaymentAgent.

DatabaseAgent.

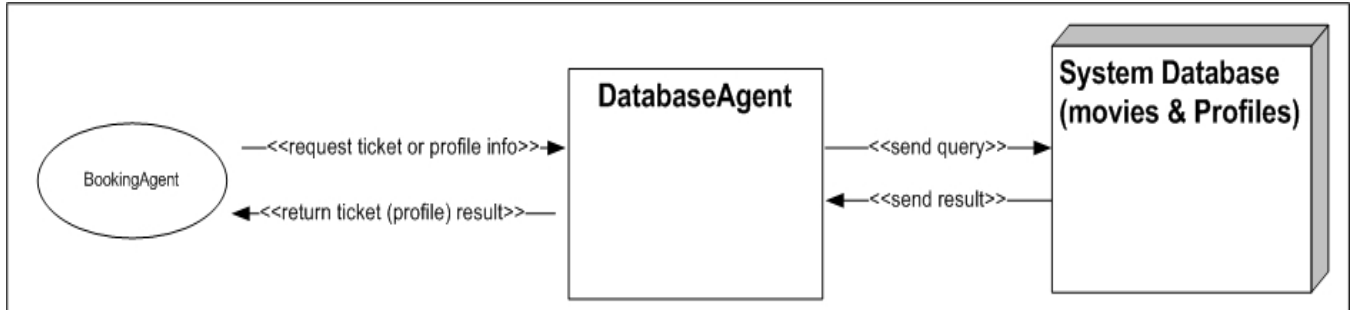


Figure 12: Use case DatabaseAgent.

Brief Description	DatabaseAgent manages the communication with the database.	
Precondition	Customer ID must be known to access the profile, or movie info for the movies.	
Postcondition	-	
Processing Step		
1	BookingAgent sends a request for a customer profile or a ticket to DatabaseAgent.	
2	DatabaseAgent processes the request and enables database access.	
3	DatabaseAgent sends database query to the database.	
4	The database returns a result for the query.	
5	DatabaseAgent returns the result of the query to the BookingAgent, either profile information or ticket information or a standard <i>no_success</i> message in case no profile exists for the customer or no ticket left.	
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.
Relationships		
Initiating	BookingAgent.	
Collaborating	BookingAgent, Database.	
Data Requirements		
Data required for PDatabaseAgent	Customer ID, movie name OR category OR number of tickets from the BookingAgent.	

Table 4: Use case description DatabaseAgent.

5.2 Inter-Agent Communication Specification

The communication architecture should be able to handle the communication between different agents, in MTD system. SOAP (Simple Object Access Protocol) provides the simplest communication system to carry out this requirement because it enables different agents with different structures to share a common communication language. It allows the communication between agents and agents and between agents and web services.

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. It 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 [6] 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 (not used here because of the extra complexity) use binary formats for their payload.

SOAP gateway translates messages between the SOAP protocol and the language of choice, therefore offering vendor, platform, and language independence. With SOAP, developers can easily bridge applications written with COM, CORBA, or Enterprise JavaBeans.

In MTD the communication encodes a HTTP header (as a web standard) and an XML message which allows us to communicate with external web services [7].

Basically a SOAP message is contained in an *envelope* which is a 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
- **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.

We will describe the input and output messages for the MTD agents based on XML specifications in the following page :

TicketRequest.

Request
<pre><TicketRequest> <movie>string</movie> <TicketDate>date</ TicketDate> <TheaterID>string</TheaterID> <NumberOfTicketsRequested>integer</NumberOfTicketsAvailable> </TicketRequest></pre>
Response
<pre><TicketResponse> <TicketID>Integer</TicketID> <movie>string</movie> <ticketDate>date</TicketDate> <TheaterID>string</TheaterID> <NumberOfTicketsAvailable>integer</NumberOfTicketsAvailable> </TicketResponse></pre>

ProfileRequest.

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

PaymentRequest.

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

5.3 Data Specification

We mention two basic data structures of the MTD system here, the movie (stored in the system database or handled by external web services) and the customer (also in database).

Movie data structure.

Field	Type	Description
MovieID	Integer	Unique identifier of the movie.
Name	String	Name of the movie.
Genre	String, enum.	Movie genre like action, drama ,....
Length	Time	Length of the movie.
TheaterID	Integer	Theater screening the movie.
Producer	String	Producer of the movie.

Table 5: Data description of movie.

Profile data structure.

Field	Type	Description
CustomerID	Integer	Unique identifier of one customer.
FName	String	First name of the customer.
LName	String	Last name of the customer.
Street	String	Street address of customer.
Town	String	Town of customer.
Province	String	Province of customer.
ZIP	String	ZIP of customer residency.
Creditcard_Number	String	Credit card number of customer.
Expiry_Date	String	Expiry date of customer's credit card.
Creditcard_Holder	String	Name of credit card holder.
Bank	String	Bank of customer.
Account_Number	String	Account number of customer.
Prefered_movies	String, enum.	Preferred movies of customer.

Table 6: Data description of user profile.

6 Conclusion

In this report we introduced the Movie Ticket Dispenser by an agent-based software development and engineering approach. We then presented the system design then an implementation overview and finally a detailed system specification.

Movie Ticket Dispenser web application as a multi-agent based ticket system enhances the performance of current ticket systems used to sell tickets. Main difference between the current ticket systems available and the proposed system is the use of a multi-agent approach in analysis and design of the system, mainly because of the benefits mentioned in chapter 2. MTD is responsible for the managing of the complete ticket purchase process, from ticket request to payment. Another difference is the access to globally available tickets. The introduced ticket system is an important step towards more intelligent and user friendly movie 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.
- [9] "SOAP Tutorial" <http://www.w3schools.com/soap/default.asp>
- [10] "Features of Web services _ SOAP" , <http://ws.apache.org/soap/features.html>