The Webbride Framework for Building Web-Based Agent Applications

Alexander Pokahr and Lars Braubach

Distributed Systems and Information Systems
Computer Science Department, University of Hamburg
{pokahr | braubach}@informatik.uni-hamburg.de

Abstract. Web applications represent an important category of applications that owe much of their popularity to the ubiquitous accessibility using standard web browsers. The complexity of web applications is steadily increasing since the inception of the Internet and the way it is perceived changes from a pure information source to a platform for applications. In order to make the task of building web applications easier many different web frameworks exist that aim at providing support for recurring and tedious development tasks. Most of the currently available web frameworks adhere to the widely accepted Model 2 design pattern that targets a clean separation of model, view and controller parts of an application in the sense of MVC. Nevertheless, existing frameworks are conceived to work with standard object-oriented business applications only and do not respect the particularities and possibilities of agent applications. Hence, in this paper a new architecture, in accordance with the Model 2 design pattern, is proposed that is able to combine the strengths of agent-based computing with web interactions. This architecture is the basis for the Jadex Webbridge framework, which enables a seamless integration of the Jadex BDI framework with state-of-the art JSP technology. The usage of web technology in combination with agents is further exemplified by an electronic bookstore case study.

1 Introduction

One key reason for the popularity of web applications is that they can be accessed via browsers in a standardized way. In this respect, they facilitate the execution of arbitrary applications without the need for installing or updating software components. These properties make web applications desirable even for more advanced and complex business tasks. Intelligent agents have been used for enterprise scale applications [2,4,11] for quite a long time. Nevertheless, few works exist that aim at a systematic integration of agent and web technology allowing to easily build web-based agent applications. Therefore, although there are many agent frameworks available few support exists on how to build agent applications employing the web as user interface. Such a setting requires anwsering some fundamental questions about how interactions should be managed between the web and the application layer and what responsibilities agents should overtake in such a scenario.

A systematic integration between both layers has the aim reducing the gap between the request/response style of browser-based interaction and the autonomous and concurrent nature of agent-based task execution. It will allow exploiting the full power of the agent paradigm for building the application logic and tie the web interface seamlessly to it. For example, in the back-end of a logistics transportation system, agents could concurrently negotiate with different subcontractors for establishing a complex multimodal transportation route while processing a single user request. In this paper we propose an architecture and a corresponding framework providing such a systematic integration and therefore allowing the efficient development of web-based agent applications.

The rest of the paper is structured as follows. In the next section an architecture for building web-based agent applications will be presented. In section 3 the realization of the architecture within the Jadex Webbridge framework is described. Thereafter, an example application will be discussed in section 4 which illustrates the usage of the web framework in a typical e-commerce scenario. In section 5 our approach will be discussed in the context of existing approaches including non-agent based frameworks as well as related agent-based solutions. Finally, we will conclude the paper with a summary and an outlook on planned future work in section 6.

2 Architecture

The goal of the approach proposed in this paper aims at seamlessly integrating agent technology and web applications. Main focus is to increase efficiency and usability for developers confronted with the task to build a web-based agent application. While interoperability with web browsers could be hand-crafted into agent applications, a generic web/agent framework allows developers to concentrate on the application problem, abstracting away from technical details.

To enable modularization and maintainability of the code-base during development, thereby also supporting specialization of developers (e.g. web engineers vs. agent programmers), the primary objective of the approach is to separate the agent-specific parts of an application from the web-specific parts such as HTML pages. Therefore, during building the web representation, the developer should not be concerned with agent-specific aspects, whereas during the development of the business logic using agents, details of the web layer should not be of great importance.

To achieve the desired independence between the web front-end and the agent application an extra layer has to be introduced, which performs the necessary mediation operations. This “glue tier” therefore allows to transparently map between details of the agent and the web layer (cf. Fig. 1). In its general form, the problem and its solutions are not specific to agent applications. As can be seen from the large number of web frameworks today (cf. section 5), a multitude of design choices exists for an implementation of mediation layers between a web front-end and some application logic. The design and implementation of the specific agent-oriented solution proposed in this paper is influenced by existing

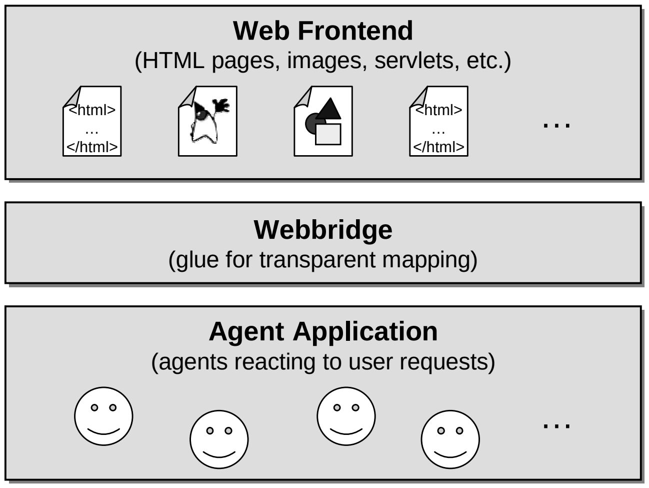
Web Frontend
(HTML pages, images, servlets, etc.)
<html>
...
</html>
<html>
...
</html>
...
Webbridge
(glue for transparent mapping)
Agent Application
(agents reacting to user requests)
...

Fig. 1. The webbridge as glue between agent and web layer

approaches and frameworks, but is motivated by the fact that the application programmer should only be concerned with the abstract and intuitive concepts of the agent paradigm. To this end, a well-established design pattern for conventional web applications is used as a starting point and is extended in a way suitable for developing agent-based applications.

2.1 Traditional Model 2 Architecture

Foundation of the proposed architecture is the widely used and accepted Model 2 design pattern [8], which adopts the Model-View-Controller (MVC, cf. [12]) approach for web development. The main idea behind this pattern is the separation of concerns, whereby each of the three proposed aspects plays a fundamentally different role. The model represents the domain-specific representation of the data on which the application operates. It is used by the view, which has the purpose to render the data in a user-friendly manner. In between, the controller serves as a connector that translates interactions with the view into actions to be performed on the model. In contrast to the MVC pattern which was conceived for desktop applications with a toolkit-based user interface, Model 2 transfers the original ideas to the web and adapts them to the request/reply-based interaction pattern. Therefore, all action in Model 2 is caused by a user that interacts with its browser and e.g. cannot arise from changes in the model data.

By separating an application into the three distinct parts application components become more manageable and can be reused or exchanged independently of each other, e.g. alternative views could be used for rendering a data model. Model 2 has been conceived by web developers who realized that it is quite difficult to use the original MVC architecture for web applications as the view

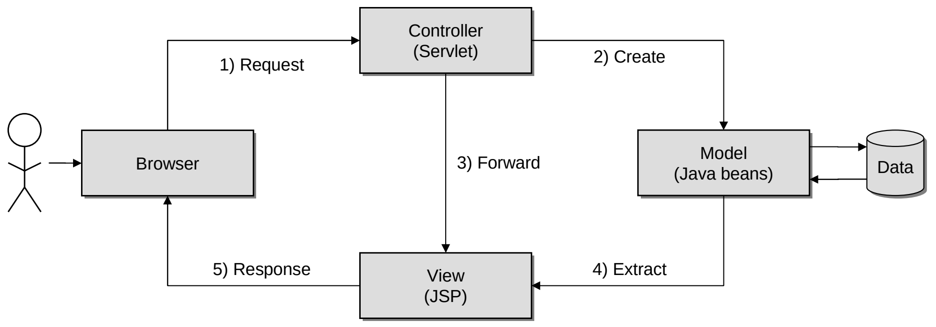

Fig. 2. Model 2 architecture (following [8])

cannot play an active role in the system. The browser always sends a request and receives a response but usually does not have the possibility to react to changes in the data model of the application.

In Model 2 the data model is kept in data bases and the data is cast into Java beans [9] for transmission and presentation. The view is typically expressed within JavaServer Pages (JSPs) [7] and the controller is represented by servlets [6]. A typical Model 2 scenario for Java web applications is depicted in Fig. 2. A browser request is issued by a user and invokes a controller servlet (1). This servlet performs the request processing, produces results in form of Java beans (2) and additionally decides which JSP to forward the request to (3). The only responsibility of the JSP is rendering the result page by utilizing the data generated from the servlet (4). The generated view is then sent back to the browser and presented to the user (5). This architecture takes advantage of the predominant strengths of both techniques, using JSP to generate the presentation layer and servlets to perform computation-intensive tasks.

As stated above the Model 2 architecture provides several benefits and allows for building complex web applications in a clean way. Additionally, its practical importance is emphasized e.g. by many non agent-based web frameworks that build on it and refine and extend its basic functionality. Hence, the direct usage of the Model 2 architecture would be beneficial but is hindered by the tight technology coupling via servlets and JSPs. In order to employ the advantages of agent technology for web-based applications modifications to the Model 2 architecture are necessary. These modifications should be carefully designed to preserve the benefits of the architecture and to enable the developer to continue using established technologies such as JSPs and JavaBeans, which have proven their value for web-based applications.

2.2 Extending Model 2 for Agents

In a web-based agent application, the agents are responsible for the execution of the application logic. In the traditional Model 2 architecture, the application

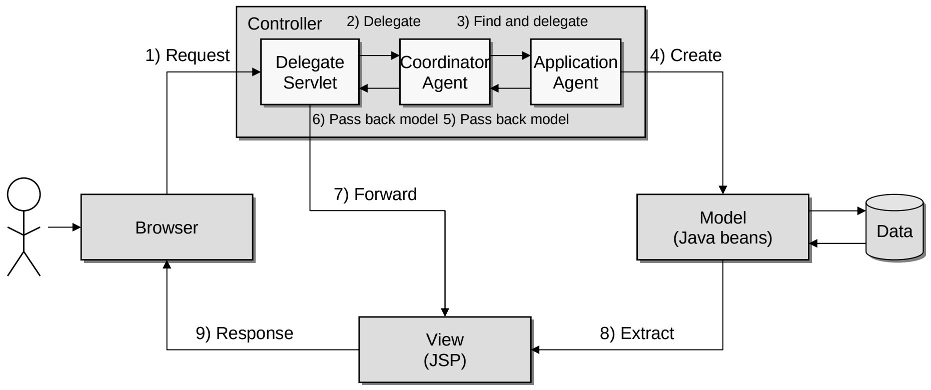

Fig. 3. Agent-based Model-2 architecture

logic is executed by the controller, which is realized as a Java servlet. To achieve the seamless integration of agents with the web, a conservative extension of the Model 2 architecture is proposed, allowing for the execution of agent behavior inside the controller. This extension allows the application functionality being designed and implemented consisting of different interacting agents. As only the controller is changed with respect to the original Model 2 architecture, the web front-end can still be realized using the well established JSP and JavaBeans technologies.

Figure 3 shows the extended Model 2 architecture proposed in this paper. To avoid application logic being scattered between the agents and the controller servlet, web requests from the browser (1) that require the execution of application logic are completely forwarded to the agent layer. Forwarding is performed in a two-step process. First the request is transferred from the *delegate servlet* to a generic *coordinator agent* (2), which acts as a mediator between the agent system and the web layer. The coordinator is responsible for finding an *application agent* that is able to process the request (3). If no suitable agent is available, the coordinator can also decide to create a new agent instance for the request. Once a suitable application agent has been identified, the coordinator sends a message to the agent, containing the details about the request. As the request is transformed to an agent message by the coordinator, the application agent does not need to know, if the request comes from the web layer or another source. After processing the request and generating the model data (4), the application agent sends the result back to the coordinator (5), which forwards it to the servlet (6). Finally, a JSP page is selected (7), which reads the results created by the application agent (8) and displays it to the user (9).

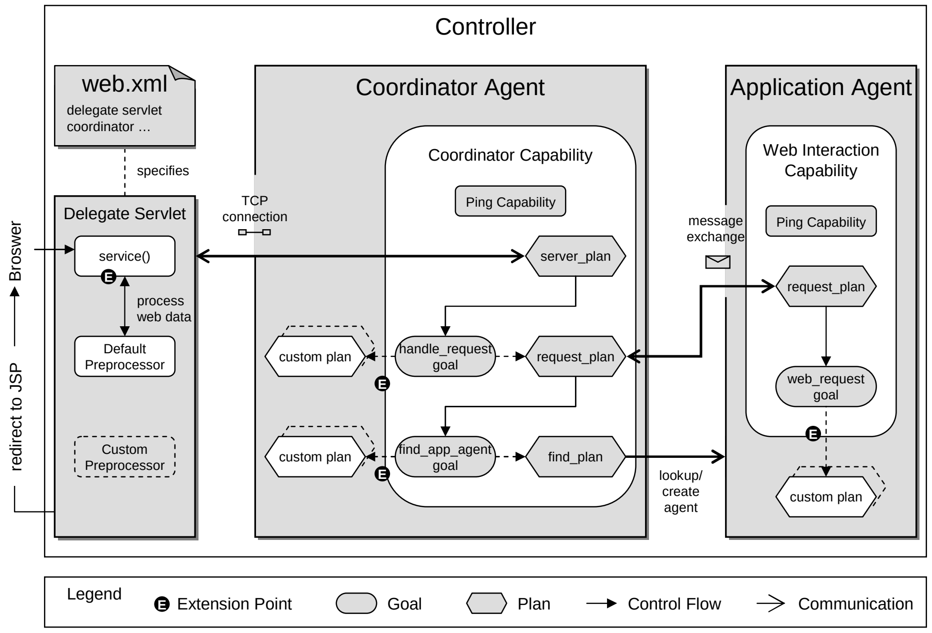

Fig. 4. Framework components

3 Framework Realization

To simplify the development of applications following the architecture presented above, a generic software framework has been developed based on the Jadex BDI (belief-desire-intention) agent system [3,16]. The framework, called Jadex Webbridge, enables application developers to focus on the three core aspects of an application, i.e. the application logic using agents, visualization via JSP pages, and the ontology-based domain data utilizing Java objects.

Following the presented architecture, the delegate servlet and the coordinator agent are responsible for mediating between these elements. Both have been realized as generic reusable components as part of the Webbridge framework. They are accompanied by a generic agent module, called web interaction capability, which can be included by the developer into application agents and handles all communication aspects with the coordinator. In the following subsections, the purpose and operation of each of these three main components of the Webbridge framework is described. The basic structure of these components and their interplay is shown in Fig. 4, which zooms further into the Controller part of Fig. 3.

3.1 Delegate Servlet

The delegate servlet has the purpose of transferring the processing of web requests to the agent layer and to finally trigger the creation of a result page. It does so by forwarding web requests to a coordinator agent using a determinate TCP connection. To tell the servlet how to establish the connection, the address of the coordinator agent can be specified in the configuration file of the web application (web.xml). After the request has been processed, the delegate servlet obtains the result from the coordinator agent and forwards the data to suitable JSP page. The application objects that are contained in the result from the application agent are copied into the forwarded request, such that they are accessible from within the selected JSP page. A default JSP page can be defined in the application configuration, but can be overridden in the result, in case different pages should be displayed depending on the outcome of the request processing.

An important responsibility of the delegate servlet is to ensure that application data (represented as Java objects) can seamlessly be exchanged between the web and the agent layer. A generic XML encoding for JavaBeans is provided, such that application objects can easily be included in the result obtained from the agent layer. On the other hand, the values of parameters in the web request are restricted to simple strings, as sent by the browser. To be able to include application objects into web pages, the framework provides an extensible pre-processing mechanism (cf. Fig. 4, left), which automatically converts string values from the web request into corresponding application objects.

3.2 Coordinator Agent

The handling of web requests in the agent layer could be based on many different strategies. The default strategy that is realized in the Webbridge framework is that the coordinator agent forwards web requests from the delegate servlet as messages to a specifc application agent instance belonging to the corresponding web session. The kind of application agent suitable for handling a specific request is defined in the web application configuration file and is therefore included in the request sent from the delegate servlet. When no agent of this kind exists for the corresponding session, a new instance is automatically created. Moreover, unused application agents are automatically removed from the agent platform after a configurable session timeout.

As different applications may have different requirements regarding the request handling, the default strategy is realized in a way, allowing it to be easily extended or adapted. The coordinator agent functionality is implemented in a reusable module called coordinator capability (cf. Fig. 4, middle). It exposes an interface in the form of goals that are created during the handling of a request. Moreover, the capability already contains default plans to handle these goals. To specify different courses of action, the developer may include the capability in a custom agent and define alternative plans for handling these goals. When the coordinator receives a request, a `handle_request` goal is automatically created, containing the details of the request. The default plan for handling this goal starts by creating another goal find_app_agent with the purpose of finding (and maybe creating) a suitable application agent. In order to check if the desired application agent already (or still) exists the default find plan uses a simple ping mechanism, which is readily available via the included Ping capabilty. After an agent has been found or newly created, the plan sends a message with the request data to this agent. The default plan for handling the find_app_agent goal realizes the session handling described above.

The two goals provide different entry points for extending or changing the request handling. By defining new plans for the find_app_agent goal, developers can realize different strategies for how a web request gets dispatched to application agents. Alternatively, by directly reacting to the handle_request goal, plans can be created to process web requests without the need for additional application agents (e.g. doing all processing in the coordinator agent or delegating to other software components). Alternative plans can be developed to handle various types of requests in different situations. For the selection of suitable plans, BDI-style reasoning is applied, such that e.g. plans get selected according to pre- or context-conditions and alternative plans are tried, when some selected plan fails.

3.3 Web Interaction Capability

Using the default strategy described in the last section, web requests are forwarded by the generic coordinator agent as messages to an application agent, which is implemented by the application developer. To simplify the development of application agents, the Webbridge framework provides a reusable module, called web interaction capability (cf. Fig. 4, right), which manages the communication with the coordinator agent. The web interaction capability is included in an application agent and automatically handles request messages sent by the coordinator. For each message a goal of type web_request is created, which has to be handled by plans, created by the application developer. The result of the goal processing is automatically communicated back to the coordinator.

From the viewpoint of application developers, the web interaction capability converts web requests into goals that belong to the application agent. Therefore, the details of the web request handling are abstracted away from agent programmers allowing them to focus on the behavior of the application agent by setting up custom plans to handle the different kinds of web_request goals.

4 Example Application

In this section, an example application developed with the Jadex Webbridge framework is described. The application represents an electronic bookstore and is inspired by the book “Developing Intelligent Agent Systems: A Practical Guide” [15]. In this scenario customers are allowed to search for and order books through a web-based user interface. Other use cases of customers of the system include

```
<web−app xmlns="http://java.sun.com/xml/ns/j2ee" version="2.4" ...>
    <servlet >
        <servlet −name>DelegateServlet</servlet−name>
        <servlet −class>jadex.bridge. delegate . DelegateServlet </servlet −class>
        <init −param>
            <param−name>agent_container</param−name>
            <param−value>localhost:9090</param−value>
        </init−param>
        <init −param>
            <param−name>session_agent_type</param−name>
            <param−value>jadex.bookstore.SalesAssistant</param−value>
        </init−param>
        <init −param>
            <param−name>webdata_preprocessor</param−name>
            <param−value>jadex.bookstore.BookstorePreprocessor</param−value>
        </init−param>
        ...
    </servlet >
    ...
    <servlet −mapping>
        <servlet −name>DelegateServlet</servlet−name>
        <url−pattern>/</url−pattern>
    </servlet −mapping>
    ...
</web−app>
```

Fig. 5. Web.xml application configuration cutout

managing their account/profile and checking the state of an order. Additionally, the system performs several back-end tasks. It has to manage a stock of books and to reorder books from wholesalers in certain intervals.

The analysis and design of the application was done using the Prometheus methodology [18] and carried further the modeling artifacts already presented in the book. The complete application logic of the system is realized as a set of collaborating agents. For the following description, especially the so-called sales assistant agent is of vital importance as it represents a personal shopping assistant for a customer. For each customer arriving at the web site an individual sales agent is created. It has the purpose of helping a customer to find and purchase books.

To show how the web/agent interaction is supported by the Webbridge framework, some code snippets from the bookstore application will be presented and explained next. The code snippets illustrate the interaction between the web frontend and the sales assistant agent.

```
public Request preprocessWebdata(HttpServletRequest request) {

    Request agentrequest    = super.preprocessWebdata(request);

    if (request.getRequestURI().indexOf("addOrderItem")!=−1) {
        int isbn    = Integer.parseInt (request.getParameter("isbn"));
        int amount = Integer.parseInt (request.getParameter("amount"));
        OrderItem  item    = new OrderItem(isbn, amount);
        agentrequest.addParameterValue("item", item);
    }
    ...

    return agentrequest;
}
```

Fig. 6. Mapping HTTP request parameters to ontology objects

4.1 Application Configuration

The basic configuration of the web related parts of the bookstore application are specified in its web.xml file. As available configuration properties are standardized [7], the application can be deployed using an arbitrary web container such as Apache Tomcat[1] or IBM WebSphere[2]. Figure 5 shows a relevant cutout of the bookstore web.xml and mainly consists of servlet descriptions and their URL-mappings. In the example, only the specification of the DelegateServlet and its mapping are shown. It uses the default DelegateServlet of the Webbridge framework (lines 2-18) and additionally defines several parameter values (lines 5-16). Among these are the contact address of the coordinator agent (lines 5-8), the class name of the application agent type (lines 9-12) and the class name of the bookstore specific webdata preprocessor (lines 13-16). In the mapping part it is defined that the DelegateServlet is the default handler for all page requests (lines 20-23). If some parts of the application should be generated by other means e.g via normal JSPs more specific mappings can be defined which have precedence over the DelegateServlet. In the bookstore example, e.g. further JSPs containing general information about the store and contact details have been defined in the full web.xml definition.

4.2 Preprocessing of Web Requests

Starting point of the scenario is that a human user is surfing at the web site of the electronic bookstore and decides to order some books after her fancy.

[1] `http://tomcat.apache.org/`
[2] `http://www.ibm.com/software/websphere`

When adding a book to the shopping cart an "addOrderItem HTTP request" is automatically generated by the browser. The request contains the item's ISBN and the amount of items to be added and is processed by the delegate servlet.

For seamless integration between the web and the agent layer, the application agent (i.e. the sales assistant agent of the bookstore) should not be required to handle details of HTTP-based interaction, such as parsing URL-patterns and MIME-encoding/decoding of request parameters. Therefore, the handling of these details is performed in the delegate servlet, which forwards only clean domain-level information based on an application-specific ontology. The mapping between data received from a web form and domain-level objects are achieved using the extensible preprocessing mechanism provided by the Webbridge framework.

E.g., the data from the "addOrderItem" web form are represented as simple strings, while the sales assistant agent only handles objects from the bookstore domain ontology containing objects such as an OrderItem. Therefore, a domain-dependent preprocessor is used by the delegate servlet to extract the values from the request (see fig. 6, lines 6, 7) and create a new domain object of type OrderItem (line 8). The ordered item is subsequently added to the agent-based request (line 9) which will be sent to the coordinator agent.

4.3 Request Execution in the Agent Layer

The coordinator agent processes the request by determining if it belongs to an ongoing conversation. In this case the request will be directly transformed into an agent message and forwarded to the corresponding application agent. Otherwise the coordinator first needs to instantiate a new application agent whose type is specified directly within the request. In this example, sales assistant agents are responsible for handling the user interaction, i.e. for each web session a corresponding sales assistant agent is created, which stays alive until the user leaves the site (as determined by a lack of activity for some time).

The agent definition file of the sales assistant is shown in Fig. 7. It includes the Webbridge functionalities via the web interaction capability (line 3). This capability mainly exports the web_request goal so that it is sufficient for the sales assistant agent to react on all domain-dependent kinds of web_request goals. In order to do this it is necessary that the web_request goal is declared and connected to the exported original one within the capability (lines 7-13). The goal exposes two in-parameters containing the domain-dependent goal type (line 9) and the agent-based web request (line 10) and one out-parameter for the agent-based response (line 11, 12).

The application code is contained in plans, which are used to process the web_request goals that are automatically created by the generic web interaction capability. The reasoning engine uses the goal parameters to find matching plans, which are executed in turn until one plan produces a suitable result. In the example, the additem_plan (lines 17-29) matches web_request goals of type addOrderItem (line 24-26). Because of the preprocessing described earlier, the agent only has to cope with application specific objects like the OrderItem (lines

```
 1  <agent name="SalesAssistant">
 2      <capabilities >
 3          <capability  name="webcap" file="WebInteraction"/>
 4      </capabilities >
 5  
 6      <goals>
 7          <achievegoal name="web_request">
 8              <assignto ref ="webcap.web_request"/>
 9              <parameter name="type" class="String"/>
10              <parameter name="request" class="jadex.bridge.onto.Request"/>
11              <parameter name="response" class="jadex.bridge.onto.Response"
12                  direction ="out"/>
13          </achievegoal>
14      </goals>
15  
16      <plans>
17          <plan name="additem_plan">
18              <parameter name="item" class="OrderItem">
19                  <value>$goal.request.getParameterValue("item")</value>
20              </parameter>
21              <body>new AddOrderItemPlan()</body>
22              <trigger>
23                  <goal ref="web_request">
24                      <parameter ref="type">
25                          <value>"addOrderItem"</value>
26                      </parameter>
27                  </goal>
28              </trigger>
29          </plan>
30      </plans>
31  </agent>
```

Fig. 7. XML definition file excerpt from the SalesAssistant agent

18-20). The plan body (omitted here) whose creation is specified within the plan head (line 21) contains the agent-based application logic to handle the customer request. One purpose of this plan is simply to update the shopping cart of the customer and store the result in the response object of the goal. In making use of the advantages of the agent-based design, the sales assistant agent further interacts with other agents in the backend of the bookstore application. It checks the availability of the item by querying a so-called stock manager agent and at the same time determines possible delivery options by negotiating with a delivery manager agent. The results of these possibly lengthy additional interactions are not passed back to the user in the context of the initial web request. Instead,

```
 1 <%@page contentType="text/html; charset=ISO−8859−1"%>
 2 <%@ page import="de.vsis.bookstore.ontology.∗" %>
 3 <% UserContext ctx = (UserContext)request.getAttribute("context"); %>
 4 <jsp:include  page="header.jsp" flush ="true"/>
 5
 6     <h1>Your Shopping Cart:</h1>
 7
 8     <% if (ctx!=null) {
 9
10         OrderItem[]  oo = ctx.getOrderItems();
11         if  (( oo!=null)&&(oo.length>0)) { %>
12
13             <a href="/bookorder?sid=<%=request.getAttribute("sid")%>">
14                 Order all   items</a>
15
16             <% for (int  j =0;j< oo.length;j ++) { %>
17                 <%=oo[j].getIsbn()%>
18                 (<%=oo[j].getAmount()+"x"+oo[j].getPrice()%> EUR)<br/>
19             <% } %>
20
21         <% }
22
23     } else  { %>
24         Your shopping cart  is  empty.
25     <% } %>
26
27 <jsp:include  page="footer.jsp"  flush ="true"/>
```

Fig. 8. JSP page for the shopping cart of a customer

they are stored locally in the beliefbase of the sales assistant, which is then able to instantly present this information to the user, if requested.

4.4 Result Page Generation

The visual part of the bookstore front-end is developed using JSP technology. To simplify the development for web programmers, the JSP pages should not have to deal with agent-related aspects of the application. Here again, the domain dependent ontology comes into play, which allows to represent all required domain data in form of JavaBeans.

The results of the processing in the agent-layer are stored by the delegate servlet directly within the original HTTP request, which is used to generate the view via a JSP. Fig. 8 shows the JSP for displaying the shopping cart (e.g. after the customer has added an item). As can be seen in lines 10-21, the information of the OderItem objects can directly be accessed from the implicit request object and is used for creating the HTML code to be presented to the user.

5 Related Work

Regarding agents and the web, there are basically two different strands of related work that need to be considered. On the one hand, a huge amount of work has been carried out in the context of traditional Model 2 Java web frameworks. In this area many different frameworks have emerged that are able to satisfy nearly any kind of developer needs. One of the first and best-known frameworks is Jakarta Struts [5], which is still widely used and also features a large developer community. Struts directly adopts the Model 2 pattern and introduces user-defined actions that perform the work of the application and finally create Java beans that can be processed in the view. Due to some limitations of Struts many fundamentally different Model 2 approaches such as Spring MVC [13] and JavaServer Faces (JSF) [10] have been proposed. A detailed comparison of many traditional web frameworks can be found e.g. in [8]. To be able to use the existing web frameworks in combination with agent technology it is necessary to embed the agents in a web framework friendly manner. This approach is e.g. followed by the Agentis AdaptivEnterprise Suite [17], which converts agents into J2EE application server components and makes them accessible for web frameworks in this way. Nevertheless, this approach limits the exploitation of agent technology as important functionalities such as the application flow and dialog management are typically handled by web frameworks cannot be delegated to the agent layer.

On the other hand, approaches need to be investigated that build up a web framework especially for agent technology and are therefore directly comparable with our architecture. Stunningly, this strand of research is nearly non-existent today. Instead, in the agent community a large body of research has been carried out in the field of interface agents aiming at the improvement of human computer interaction e.g. [14] but this does not directly contribute to the problem addressed in this paper. The only generic approach is provided by the JACK WebBot solution[3] which can be used to equip JACK agent applications with a web front-end. The approach is similar to ours as also the controller part represents the mediator component between the web and the agent layer. Although the WebBot architecture is very flexible, it does not provide a clean framework approach. Instead, the agent programmer has to design and implement generic functionalities such as agent session management by herself and cannot make use of predefined modules for that purpose. Additionally, it does not allow consistently using the same ontology objects on all tiers and hence requires tedious conversions being done by the application instead of the framework.

Besides the WebBot architecture, also some ad-hoc solutions exist, which use external interfaces provided by an agent platform (e.g. the JadeGateway class in JADE [1] or the HabitatGateway class in Tryllian's ADK[4]). As such interfaces

[3] `http://www.agent-software.com`

[4] `http://www.tryllian.com`

only provide generic access to the agent platform, most of the technical details concerning the connection of agents with the web layer have to be handcrafted by the developers in these approaches.

6 Conclusion and Outlook

This paper has presented an architecture and a framework simplifying the development of web-based agent applications as these kinds of systems gain steadily more importance in the context of business solutions. To achieve an integration between the web and the agent world a novel agent-based architecture conformant to the well-known Model-2 design pattern has been proposed. The agentified Model 2 architecture intentionally refines only a small part of the original architecture by further developing the *controller component*. This enables using agents for all aspects related to application functionality while preserving the usage of the existing and well suited Model 2 techniques for rendering (JSPs) and model representation (JavaBeans). One crucial aspect of this extended architecture is the partitioning of the controller into three distinct functionalities: *delegate servlet*, *coordinator agent* and *application agents*. The delegate has the main purpose to forward business tasks that originate from browser requests to the coordinator agent. The coordinator processes requests by distributing them to domain-dependent application agents. A main advantage of the proposed generic architecture consists in the separation of concerns established by Model 2. The architecture therefore detaches cleanly the web layer from the agent layer facilitating their largely independent development.

Moreover, the Jadex Webbridge framework implementing the aforementioned architecture has been presented. The main characteristic of this framework is the support for agent technology in the context of web applications. The framework provides ready-to-use and extensible functionalities realizing the delegate servlet and the coordinator agent. Additionally a web interaction module (capability) is provided that encapsulates the generic functionalities needed by application agents. This capability transfers web requests to web_request goals which can be handled in the same way as any other ordinary agent goal. The capability automatically handles all interactions with the coordinator and reduces the task of the agent developer to writing plans for the domain logic of pursuing web_request goals.

Future work will be targeted at improving the processing of web interactions. Currently, web interactions are short-lived meaning that request goals are created whenever a user issues a new browser request so that the interaction state has to be preserved within the agents beliefs. A more advanced approach would allow to treat a conversation as a whole e.g. within a plan allowing the agent to manage the interaction in a similar sense as normal message-based protocols. This would mean that not only the functionality of one short-term interaction goal could be captured in a BDI plan, but a whole workflow (e.g. the book buying use case in the example presented).

References

1. F. Bellifemine, G. Caire, and D. Greenwood. *Developing Multi-Agent systems with JADE*. John Wiley & Sons, 2007.
2. S. Benfield, J. Hendrickson, and D. Galanti. Making a strong business case for multiagent technology. In *5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006)*, pages 10–15, New York, NY, USA, 2006. ACM Press.
3. L. Braubach, A. Pokahr, and W. Lamersdorf. Jadex: A BDI Agent System Combining Middleware and Reasoning. In R. Unland, M. Calisti, and M. Klusch, editors, *Software Agent-Based Applications, Platforms and Development Kits*, pages 143–168. Birkhäuser, 2005.
4. J. Castro, M. Kolp, and J. Mylopoulos. Developing agent-oriented information systems for the enterprise. In *Proceedings of the Second International Conference on Enterprise Information Systems (ICEIS 2000)*, pages 9–24, Escola Superior de Tecnologia de Setúbal / Campus do IPS, 2000. ICEIS Secretariat.
5. C. Cavaness. *Programming Jakarta Struts*. O'Reilly Media, 2004.
6. D. Coward. *Java Servlet, Specification Version 2.3*. Sun Mircosystems, 2001.
7. P. Delisle, J. Luehe, and M. Roth. *JavaServer Pages, Specification Version 2.1*. Sun Mircosystems, 2006.
8. N. Ford. *Art of Java Web development: Struts, Tapestry, Commons, Velocity, JUnit, Axis, Cocoon, InternetBeans, WebWorks*. Manning Publications, 2003.
9. G. Hamilton. *JavaBeans, Specification Version 1.01*. Sun Mircosystems, 1997.
10. J. Holmes and C. Schalk. *JavaServer Faces: The Complete Reference*. McGraw-Hill Osborne Media, 2006.
11. N. R. Jennings and M. J. Wooldridge. *Agent Technology - Foundations, Applications and Markets*. Springer, 1998.
12. G. Krasner and S. Pope. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of Object Oriented Programming*, 1(3):26–49, 1988.
13. S. Ladd, D. Davison, S. Devijver, and C. Yates. *Expert Spring MVC and Web Flow*. APress, 2006.
14. P. Maes. Agents that reduce work and information overload. *Communications of the ACM*, 37(7):30–40, 1994.
15. L. Padgham and M. Winikoff. *Developing Intelligent Agent Systems: A Practical Guide*. John Wiley & Sons, 2004.
16. A. Pokahr, L. Braubach, and W. Lamersdorf. Jadex: A BDI Reasoning Engine. In R. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni, editors, *Multi-Agent Programming: Languages, Platforms and Applications*, pages 149–174. Springer, 2005.
17. P. Taylor, P. Evans-Greenwood, and J. Odell. Agents in the enterprise. In *Proceedings of the Australian Software Engineering Conference (ASWEC 2005)*, pages 9–24. IEEE Computer Society, 2005.
18. M. Winikoff and L. Padgham. The Prometheus Methodology. In F. Bergenti, M.-P. Gleizes, and F. Zambonelli, editors, *Methodologies and Software Engineering For Agent Systems*, pages 217–234. Kluwer Academic Publishers, 2004.