Proceedings of the 5th WSEAS International Conference on Applied Computer Science, Hangzhou, China, April 16-18, 2006 (pp312-317)

Agent-Enabling Transformation of E-Commerce Portals with Web Services

David B. Ulmer
CTO
Sotheby's
New York, NY 10021, USA

Lixin Tao
Professor
Pace University
Pleasantville, NY 10570, USA

Abstract: - Web services and middleware technologies are adopted to provide bi-directional communication channels between network-blind generic transaction agents and existing e-commerce portals with no modification to their existing functions. An action-word-based translation system is used to bridge the semantic gap between the agents and the e-business portals.

Key-Words: - E-commerce portals, transaction software agent, Web service, middleware, transaction automation

1 Introduction

Given the ubiquitous nature of electronic marketplaces on the Internet, buyers have an enormous variety of items available to them, effectively overwhelming them with choices. A new e-commerce paradigm is in high demand in which software agents can play an important role in automating many transactional activities like the discovery, comparison, selection, purchase and shipping of products. But so far, the Web-based marketplaces have been mainly designed for human interaction and do not support efficient interaction with software agents.

This paper studies the special needs of transaction agents, enhances the current e-commerce software architecture for transaction agent support, and designs the corresponding interfaces and supporting systems. The interface design and supporting systems will accommodate the format and interface needs of the human users, while enhancing the transactional capabilities of software agents through an adaptable and robust interface. Transaction agents can operate in either a pull or push manner, which require different supporting systems and architecture. Web services are utilized that facilitate an *Agent-Enabling Interface (AEI)* API for exposing the business logic of an e-commerce portal by leveraging existing Internet protocols. The client-side support system features a *Generic Middleware between Agents and Portals (GMAP)* that encapsulates the transaction agent and minimizes its modification to enable e-commerce portal interoperability.

2 Problem Statement

Several assumptions are made in the research to limit the scope of the problem and solution. The main assumptions are:

- A transaction agent is a standalone application running on the client side. Its internal design and implementation are hidden from its environment (i.e., a black-box). An agent communicates with its environment through tables of columns of data, and the meaning of each such column is described by a sequence of keywords.
- The transaction agent is generic in nature, meaning they are designed to work with a broad range of portals and are not specifically designed for a particular portal.
- Using the input criteria and internal logical reasoning capability, a transaction agent can generate the sub-transactions (steps that can be mechanically translated and executed by its supporting system) incrementally based on feedbacks from the portals or service registries through the client-side support system.
- A transaction agent specifies portal operations in a set of standardized action words based on the *Consumer Buying Behavior (CBB)* model [2], which comprises the actions and decisions involved in buying goods and services online.
- The solution is based on the Java 2 Enterprise Edition™ Platform Specification although the concepts can apply to other application frameworks as well.

The design objectives for this research include:

1. Providing efficient e-commerce portal interfaces for both browser-based human clients and client-side software applications including transaction agents or their support systems.

2. Supporting generic transaction agent and portal interoperability through compatible client and server-side support systems.
3. Supporting both pull and push modes of agent-portal interactions to facilitate a dynamic and event-driven business transaction environment.
4. Utilizing open standards and technologies to accommodate a broad range of e-commerce portals, transaction agents, and client platforms.
5. Minimizing changes to the software agent design, but more importantly avoiding changes to the back-end business logics and database structure of the e-commerce portals.
6. Adopting component-based system design and maximizing reuse through generic functions.
7. Avoiding software installation for an agent to interact with a particular e-commerce portal.

3 Remote System Integration with Web services

To achieve our design objectives, we must first choose an appropriate technology to support robust platform-independent system integration between client-side transaction agents, which are usually network-blind, and the e-commerce portals.

Web services are a new approach for integrating systems on distributed heterogeneous platforms. It is based on open standards for XML, XML Schema, SOAP, WSDL, and UDDI [3]. It has the following advantages over the traditional technologies for system integration, including CORBA and COM+:

- It supports client/server interaction with HTTP, a widely accepted Web protocol. Invocation on remote methods is through the standard HTTP GET/POST requests on port 80, thus avoiding the firewall configuration problem for system security. This is a very desirable feature for agent-portal communications.
- Web service is fundamentally a wrapper technique. It can be used to wrap up an existing legacy application rapidly and expose its business logics on the Internet for remote clients to access. Therefore it is the appropriate technology for adding an agent API to an existing portal, no matter what platform or software framework the portal is based on, with minimal changes to the portal.
- Since XML supporting tools are publicly available and have already been integrated into many client platforms like Java, Microsoft Windows, and Unix/Linux, Web service clients don't need a service-specific software installation for communicating with a particular Web service provider. A WSDL file describing a service provider's service will be downloaded and used to create client-side proxies for accessing remote services. Since a generic transaction agent is supposed to interact with a large variety and number of e-commerce portals, avoiding a client-side software installation for each portal service is critically important.
- As part of the core Web service technologies, UDDI supports service registration and universal discovery based on service descriptions. A UDDI registry could function like a business yellow book for an agent's supporting system to browse and search for suitable portals to fulfill a client's transaction need.

To create a Web service provider, business logic methods are first identified, and a tool is used to generate a WSDL (XML) file describing the types and method signatures of the Web services. Server components are also generated including a servlet (Agent Interface Servlet) as a service call entry point and a service tie object that can call the local business logic methods on behalf of remote clients. The WSDL file is registered with a UDDI registry under specific categories.

For a client to obtain a service, it will first search a UDDI registry to identify one of the potential Web service providers and download its WSDL file. A tool will be used to generate from the WSDL file the native proxy source files for the portal's Web services. To call a server-side business logic method, the client will call the same method on its local proxy object, which will wrap the invocation in a SOAP (XML) message and send it to the service provider's entry point with an HTTP request. The receiving servlet forwards the SOAP message to the tie object to generate the local invocation on the business logic method. The response from the method is converted by the tie object into a SOAP message and forwarded to the servlet for returning back to the proxy as the response to its HTTP request. The client-side proxy will convert the response SOAP message into the actual response data and forward it to the client as its own method return value.

In this research we use Web services as the fundamental technology to integrate client-side transaction agents with e-commerce portals.

4 Exposing the Portal's Business Logics

Exposing the business API of the portal is essentially a translation problem in which the protocol understood by the transaction agent must be converted to the protocol of the business API for the

portal. This translation problem can be solved by designing software to act as an in-between layer that performs the protocol conversion between the two entities. This is a well understood problem solvable with a middleware architectural pattern (see Figure 1).

Given the distributed nature of the problem, the middleware layer must be split onto both the client and server tiers and act in a plug compatible fashion (able to interface to each other by design). For this research, the middleware is represented by the client and server-side generic *Agent Interface Support Systems* (see Figure 2).

As explained in the previous section, Web services are beneficial and facilitate the *Agent-Enabling Interface (AEI)* for exposing the business logics of an e-commerce portal by leveraging existing Internet protocols. We assume that the e-commerce portals that adopt our approach for supporting remote transaction interaction are based on the tiered software architecture (a best and now common practice). The presentation tier is mainly a servlet container containing multiple servlets or JSP pages for processing HTTP requests. The business logics are encapsulated in the application server tier consisting of an EJB container with multiple EJBs providing a scalable business logic implementation and database rows' in-memory caching. The database tier uses relational databases to provide data persistency. The approach described in this paper can also be easily adapted to support portals based on Microsoft's .NET platform.

For such a portal to adopt our approach and expose its selected business logics to the remote transaction agents, the following major steps should be taken:

1. The business logic methods to be exposed to the public remote transaction agents should be identified. Normally these are EJB methods invoked from the presentation tier. If we expose all the EJB methods used by the presentation logics to the remote agents, the agents will have access to the same business logics as the human clients do through Web browsers.
2. With the signatures of these selected business logic methods as input, a generic Web service tool will be used to generate a WSDL (Web Services Description Language) file describing the connection entry point, types and method signatures; a servlet functioning as the Web service entry point; the tie source file for converting between SOAP messages and business logic method invocations; and other related supporting resources.
3. The WSDL file will be registered with public UDDI registries under proper industry categories.
4. A portal-specific mapping table is designed based on our *Action Word Mapping* class, which allows the portal designer to associate the action words, standardized for transaction agents to specify generic transaction operations based on the CBB model [2], with business logic methods. The table will also provide keyword descriptions for all method signatures, including those for the meaning of each of the parameters and the meaning and getter method for each of the value components of a returned object. This is critical to resolving the semantic gap between the transaction agent and the portal's business logic methods. A getter method for such an *Action Word Mapping* object will also be exposed as one of the Web service methods.
5. As soon as the entry point servlet is deployed in the presentation tier servlet container, the portal will have the selected business logic methods exposed for transaction agents to access in pull mode. We also added a *Callback* tier along with the portal presentation tier to provide transaction agents with push mode access to the portal business logics [5].

It can be observed that our approach of exposing business logics to remote transaction agents in pull mode requires no modification to the existing portal design and deployment, and no software development.

5 Generic Middleware between Agents and Portals

The client-side support system is the client-side half of the middleware. It is responsible for enabling network-blind transaction agents to interact with both Web service registries to search for suitable e-commerce portals and the business logic methods of the portals.

This research proposes a reusable generic client-side agent support system called *Generic Middleware between Agents and Portals (GMAP)*. GMAP provides the infrastructure and implementation for the major generic service functions including the *Registry Query Manager*, the *Portal Invocation Manager* and the *Callback Web Service*. This section only focuses on the GMAP components necessary to support the pull-mode access to remote portals (Figure 3).

After a human client passes his/her transaction specifications to a transaction agent through an input table (establishing an unmet need as a business transaction), the agent will start to issue

sub-transactions consisting of action words and their appropriate arguments to carry out the transaction. These action words and arguments are passed to GMAP through a *Sub-transaction* table. Further sub-transactions may be generated incrementally based on feedback from the UDDI registries or e-commerce portals through GMAP.

Typically an agent-assisted transaction will start with searching for suitable e-commerce portals from public UDDI registries. The Registry Query Manager is responsible for this task. It will use keywords derived from the transaction specification parsed by the agent to download information for all suitable portals, which include description keywords and service description WSDL files, rank them according to the degree that their own descriptions match that of the specification keywords, and pass the result back to the agent through a *Service Definition table* for further selection. Upon retrieving the service descriptions, the Registry Query Manager uses a generic Web service tool to transform the portal's WSDL file into a client-platform-dependent proxy source file for that portal's Web services, compile it, and generate a proxy instance inside GMAP exposing exactly the same interface as the business logic methods exposed on that portal.

A *Personal Information Manager* will provide a persistent cache through a database for personal information of a human client, including his/her authentication information for selected portals, his/her shipping and billing addresses, his/her credit card or bank account information, and the information of his/her frequently adopted portal services. This persistent cache supports the learning capability of GMAP to better serve the needs of individual human clients.

Upon the selection of a particular e-commerce portal for further exploration for a transaction, the *Portal Invocation Manager* will first call a method, the signature of which has been standardized by our interface design, against the proxy object to download from the portal its unique *Action Word Mapping* object for closing the semantic gap between the transaction agent and the portal. The Portal Invocation Manager will maintain the proxy object and the Action Word Mapping object for the duration of the transaction. The information in both the WSDL file and the downloaded Action Word Mapping object will be used to fill up a *Method Description* table for the Web service methods of that portal. In this table, each method is mapped to one or more action words standardized for the e-business domain, and its signature is further described by keywords carefully chosen by the portal designers. This table is critical for the transaction agent to help select the right business methods to invoke with the right arguments.

Each time the agent or GMAP needs to invoke a remote business logic method of a portal, the Portal Invocation Manager will call the same method against the portal's proxy object local to GMAP. The conversion from a string form of method name and arguments, provided by the agent, to the actual method invocation is implemented by Java's reflection framework. This is a key feature that makes our GMAP a truly generic reusable software component since it has no hard-coded method calls to any portal-specific Web services.

Upon receiving a method call, the proxy object's method body will convert the invocation (method name and arguments) into a SOAP message, and send it to the corresponding portal's Web service entry-point servlet as the entity body of an HTTP request. The HTTP response will return another SOAP message representing the return object or exception information of the portal's business logic method. The proxy's method body will convert the return SOAP message into a return object of a type defined by and generated from the port's WSDL file, and return the object as its own. The Portal Invocation Manager will use the information in the Action Word Mapping object to identify the getter methods of the returned object, and use the getters to retrieve the returned value components, populate them into a result table, and qualify each of these value components with description keywords contained in the Action Word Mapping object. When all the return values for the current sub-transactions have been populated into the result table, the table will be passed to the transaction agent for further processing [5].

References:

[1]Berners-Lee, T., Hendler, J., Lassila, O., "The Semantic Web," *Scientific American,* 284(5) 34-43, May 17, 2001.

[2]Guttman, R., Moukas, A., Maes, P., "Agent-mediated Electronic Commerce: A Survey," *Knowledge Engineering Review,* 13(2), pp. 143-152, June 1998.

[3]Sun Microsystems, "The Java Web Services Tutorial for JWSDP v1.6," July 14, 2005, URL: http://java.sun.com/Webservices/tutorial.html, (valid by March 3, 2006).

[4]UDDI Spec TC Committee, "UDDI v2 Specifications", http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv2, (valid by March 3, 2006)

[5]Ulmer, D., "Architectural Solutions to Agent-Enabling E-Commerce Portals with Pull/Push Abilities," doctoral dissertation, Pace University, 2004.

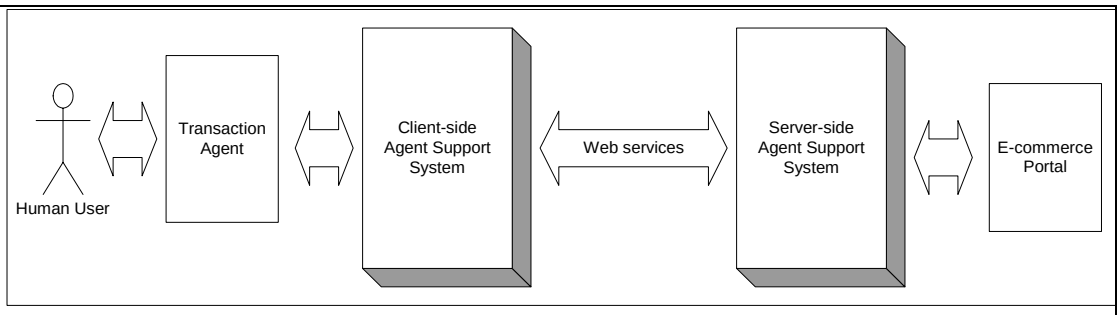

Figure 1 – Middleware -- Agent Support Systems

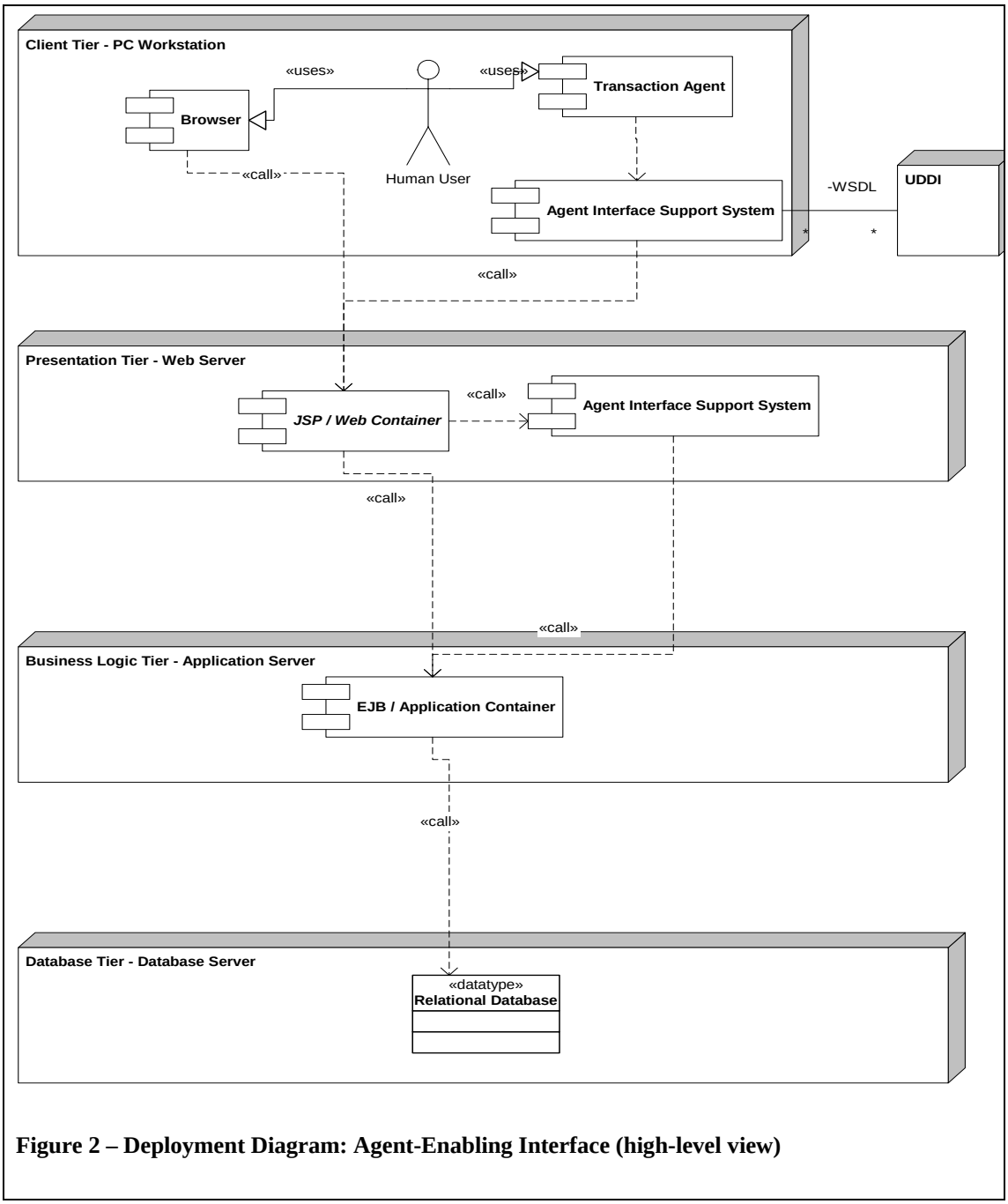

Figure 2 – Deployment Diagram: Agent-Enabling Interface (high-level view)

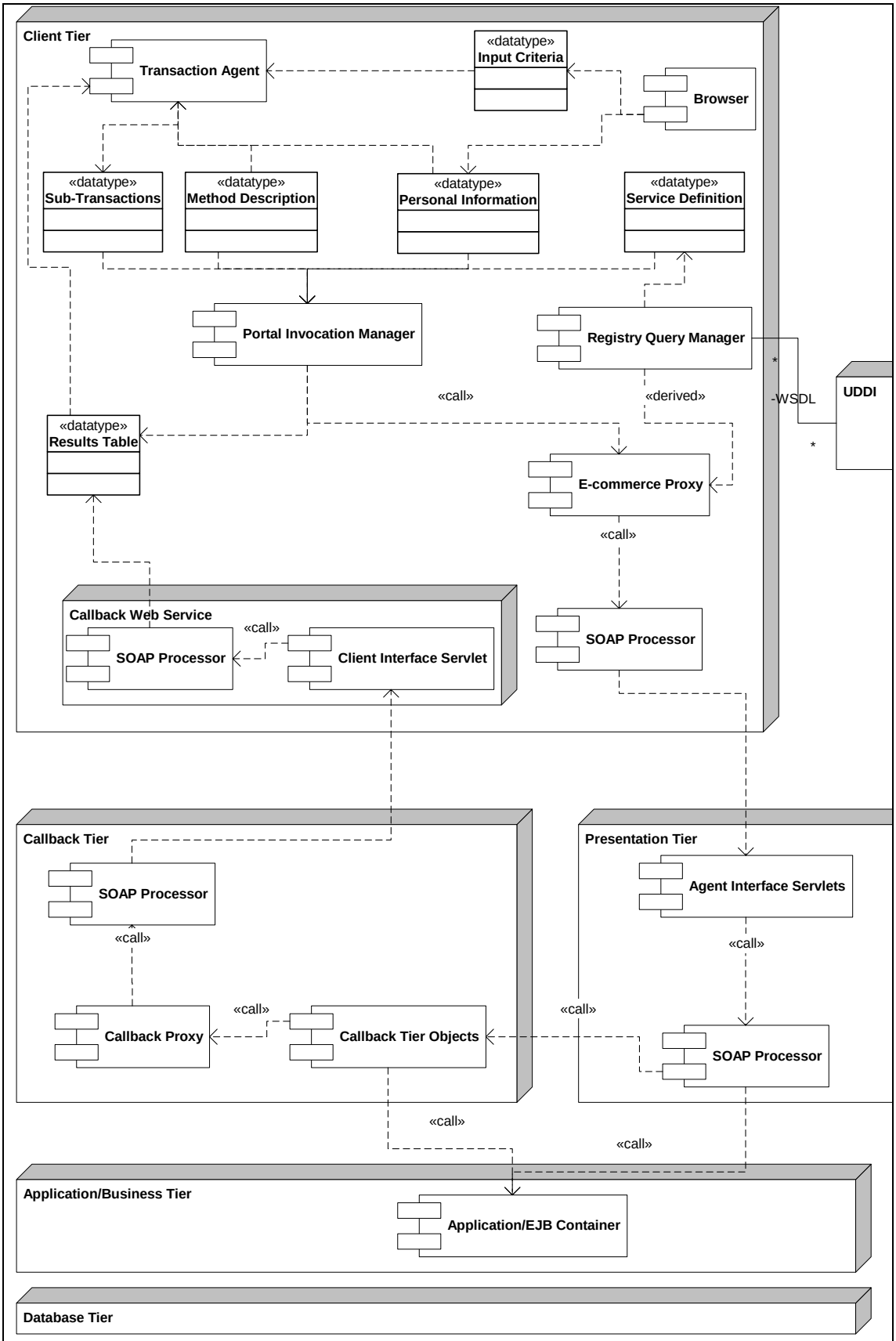

Figure 3 – Component Diagram: Detailed View