Architectural Solutions to Agent-Enabling E-Commerce Portals with Pull/Push Abilities

By

David B. Ulmer, B.A., M.B.A, M.S., M.S.

Submitted in partial fulfillment
of the requirements for the degree of
Doctor of Professional Studies
in Computing

at

School of Computer Science and Information Systems

Pace University

March 2004

Copyright

Approval Page

We hereby certify that this dissertation, submitted by David B. Ulmer, satisfies the dissertation requirements for the degree of Doctor of Professional Studies in Computing and has been approved.

____-_______________
Lixin Tao Date
Chairperson of Dissertation Committee

____-_______________
Fred Grossman Date
Dissertation Committee Member

____-_______________
Michael Gargano Date
Dissertation Committee Member

School of Computer Science and Information Systems
Pace University 2004

Abstract

Architectural Solutions to Agent-Enabling E-Commerce Portals with Pull/Push Abilities

by
David B. Ulmer

Submitted in partial fulfillment
of the requirements for the degree of
Doctor of Professional Studies
in Computing

March 2004

The Internet-based World Wide Web has had an enormous impact on business and society. It has succeeded largely because of its open architecture and ease-of-use. Although it was originally designed for distributed and interactive information sharing, it has evolved into a powerful business platform in which electronic commerce is driving fundamental change in consumer buying patterns. However, because of 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 of products or services, and the shipping of products. But so far, the Web-based marketplaces are mainly designed for human interaction and do not support efficient interaction with software agents.

This dissertation studies the current e-commerce software architecture and the special needs of transaction software agents, enhances the current e-commerce software architecture for 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. This design is further enhanced to enable the transaction agent to react to marketplace changes and delayed transactions via callback from the server, implemented through a deviation from the current software architecture with a server-side Callback tier. The client-side supporting system is also proposed and designed to work with generic transaction agents. It features a Generic Middleware between Agents and Portals (GMAP) that insulates the transaction agent and minimizes its modification to enable e-commerce portal interoperability. The supporting systems on the client and server communicate via Web services and provide the functionality required to enable a

transaction agent to address and call the business logic of an e-commerce portal thus preserving the back-end of the portal.

In summary, this research defines solutions that improve the interoperability between transaction software agents and e-commerce portals via client and server supporting systems. Using an exposed business API, this enhancement enables agents to execute the same business logic as the human-based interface thus eliminating changes to the back-end of the portal to support transaction agents.

Acknowledgements

Achieving the goal of earning a doctorate degree has been a life long desire that brings me great personal satisfaction. But as with most good things in life, I could not have done it alone.

First, I would like to thank my advisor, Dr. Lixin Tao whose wonderful counsel and tireless encouragement provided me the insight and help I needed. His keen intellect, leadership and experience inspired me to go further than I thought I could. I will always be grateful to him for the mentoring he provided me throughout this process. I would also like to thank my other committee members, Dr. Michael Gargano and Dr. Fred Grossman, for their comments and suggestions. Dr. Grossman in particular motivated me early on regarding my research idea, and I thank him for the encouragement he gave me and for his leadership of the D.P.S. program.

Also, I would like to thank my wonderful family. First, to the memory of my mother, Ellen, who instilled the importance of education into my life. She provided the spark for me to succeed in this endeavor, and her guiding light illuminated the path for this dream come true. To my immediate family, whose support during the toughest moments of balancing personal, professional and school responsibilities helped ease the burden on me. To my special mother and father in-law, Ann and William, they have been and continue to be a very positive and supportive influence in my life. To my lovely daughters, Marisa, Jaclyn, and Lora, may you pursue excellence and achieve your own aspirations and desires. Learning is a life long pursuit and I hope you find it as rewarding as your mother and I do. Lastly, to my precious wife, Marcia, whose constant love and support was always there for me. You led the way by achieving your dream, and I'm so very proud of you for being the <u>first</u> Dr. Ulmer.

Table of Contents

Approval Page ii
Abstract iii
Acknowledgements v
Table of Contents vi
List of Tables viii
List of Figures ix
Chapter 1 – Introduction 10
1.1 Complexity of a Typical E-Commerce Transaction 11
1.2 Software Agents' Potential Role in E-Commerce Transaction Automation 13
1.3 Serving Human Users and Software Agents: A Great Challenge 17
1.4 Solution Strategy 19
1.5 Major Contributions 20
1.6 Dissertation Organization 21
Chapter 2 – E-Commerce Portals, Transaction Agents and System Integration 23
2.1 Contemporary Web Architecture and Its Limitations 23
2.2 Transaction Agents and their Necessary Supporting Environments 28
2.3 Remote System Integration across Heterogeneous Platforms 37
2.3.1 Common Object Request Broker Architecture (CORBA) 38
2.3.2 Java Remote Method Invocation 40
2.3.3 Microsoft .NET 41
2.3.4 Web Services 42
Chapter 3 – Solution Strategies 49
3.1 Problem Statement 49
3.1.1 Principles 51
3.1.2 Objectives 52
3.1.3 Assumptions 52
3.2 Semantics between Generic Transaction Agents and Portals 55
3.3 Contemporary E-Commerce Portal Design 58
3.3.1 E-Commerce Use Cases 58
3.3.2 Software Architecture Overview 61
3.3.3 High-Level System Landscape 63
3.4 Exposing Open Business APIs with Web Services 64
3.4.1 Why Web services 66
3.4.2 Identifying Business Methods for Exporting 68
3.4.3 The Agent-Enabling Interface – Conceptual Design 69
3.4.4 Generating WSDL files 70
3.4.5 Generating Web Services 71
3.4.6 Registering Web Services in Public Registries 73
3.4.7 The Agent-Enabling Interface – Components Description 74
3.5 Supporting Delayed Transactions with Server Callbacks 77
3.5.1 Value of Delayed Transactions 78
3.5.2 Server Callback: Unsupported by Web Service Technologies 79
3.5.3 Enhancing the Contemporary Web Architecture with Callbacks 80
3.6 Summary 83

Chapter 4 – Enabling Portals to Work with Pull/Push Agents 84
4.1 Generic Functions Supporting Pull/Push Agents... 84
4.2 Assumptions for the Generic Callback Tier Design 85
4.3 Callback Tier: A New Generic Web Application Tier for Push Services 87
4.3.1 Registration 88
4.3.2 Scheduling.. 89
4.3.3 Checking 89
4.3.4 Client Callback... 90
4.4 Callback Tier: Application Programming Interface (API) 90
4.4.1 Registration Interface 90
4.4.2 Scheduling Method 95
4.5 A Logical Design of the Callback Tier 96
4.5.1 In-Memory Callback Table... 97
4.5.2 In-Memory Scheduling Table 98
4.6 Integration Strategies of Callback with Legacy Portals.................................... 98
4.7 Typical Use Cases and Workflow: Push Mode 99
4.8 Summary 103
Chapter 5 – A Generic Client-Side Agent Interface Support System............................ 105
5.1 Interface Abstraction of Transaction Agents 105
5.2 GMAP: Generic Middleware between Agents and Portals 107
5.2.1 Registry Query Manager (RQM) 108
5.2.2 Portal Invocation Manager (PIM)... 108
5.2.3 Callback Web Service 110
5.2.4 E-Commerce Proxy... 110
5.3 A Logical Design of GMAP 112
5.3.1 Input Criteria Table... 113
5.3.2 Sub-transactions Table.. 113
5.3.3 Callback Registration Table.. 115
5.3.4 Service Definition Table 116
5.3.5 Method Description Table 117
5.3.6 Results Table... 119
5.4 Integration of Transaction Agents with GMAP.. 120
5.5 Typical Use Cases and Workflow: Pull Mode.. 122
5.6 Summary 126
Chapter 6 – Conclusion.. 128
6.1 Major Contributions.. 128
6.2 Future Work 131
References... 132

List of Tables

TABLE 1 – ONLINE SHOPPING FRAMEWORK WITH EXAMPLES OF AGENT MEDIATION 31

TABLE 2 – ACTION WORDS 58

TABLE 3 – IN-MEMORY CALLBACK TABLE 98

TABLE 4 – IN-MEMORY SCHEDULE TABLE 98

TABLE 5 – DELAYED TRANSACTIONS WORKFLOW 102

TABLE 6 – INPUT CRITERIA TABLE 113

TABLE 7 – SUB-TRANSACTIONS TABLE 114

TABLE 8 – CALLBACK REGISTRATION TABLE 116

TABLE 9 – SERVICE DEFINITION TABLE 117

TABLE 10 – METHOD DESCRIPTION TABLE 117

TABLE 11 – RESULTS TABLE 120

TABLE 12 – GMAP WORKFLOW 125

List of Figures

FIGURE 1 – USE CASE: CONTEXT LEVEL E-COMMERCE PORTAL 59

FIGURE 2 – USE CASE: E-COMMERCE PORTAL 60

FIGURE 3 – DEPLOYMENT DIAGRAM: E-COMMERCE SYSTEM LANDSCAPE 64

FIGURE 4 – MIDDLEWARE: AGENT SUPPORT SYSTEMS 65

FIGURE 5 – DEPLOYMENT DIAGRAM: AGENT-ENABLING INTERFACE (HIGH-LEVEL VIEW) 69

FIGURE 6 – COMPONENT DIAGRAM: AEI (CLIENT-SIDE DETAIL) 77

FIGURE 7 – COMPONENT DIAGRAM: AEI (SERVER-SIDE DETAIL) 82

FIGURE 8 – REGISTRATION INTERFACE 91

FIGURE 9 – CALLBACKCRITERIA CLASS 93

FIGURE 10 – COMPONENT DIAGRAM: CALLBACK TIER 97

FIGURE 11 – ACTIVITY DIAGRAM: CALLBACK TIER TRANSACTION WORKFLOW 103

FIGURE 12 – SEQUENCE DIAGRAM: GMAP 107

FIGURE 13 – COMPONENT DIAGRAM: GMAP 112

FIGURE 14 – USE CASE: PURCHASE SCENARIO 123

FIGURE 15 – ACTIVITY DIAGRAM: GMAP WORKFLOW 126

Chapter 1 – Introduction

Since 1999, the number of e-commerce portals has exploded on the World Wide Web (WWW). The IDC predicts global electronic commerce to exceed $1 trillion by 2003 [14]. Given the ubiquitous nature of electronic marketplaces on the Internet, buyers have an enormous number of choices available to them. The challenge this presents to the user is one of breadth and depth; breadth in terms of choices between the items of interest and depth as it relates to the number of e-commerce portals to search and transact with. In order for users to have an optimal electronic commerce experience (meaning buying the best products at the best price across multiple venues), they need the ability to deal with the large volume and continuing growth of items and options. Consumers require additional automation in finding, choosing, purchasing and shipping the items to help transact business with e-commerce portals.

With this growing popularity of electronic commerce, a new paradigm for doing business on the Internet is needed. Guttman et al. [16] state "generally, the more time and money that can be saved through automation, the easier it is to express preferences, the lesser the risks of making sub-optimal transaction decisions, and the greater the loss for missed opportunities, the more appropriate it is to employ agent technologies in electronic commerce." Automation (i.e., agents) to improve the efficiency and quality of decisions may be an important need for the continued expansion of the e-commerce marketplace.

The premise is that in the decade ahead, the global economy and Internet will become an amalgamated information economy resulting in billions of electronic commerce opportunities, thus requiring the automation of efficient and effective software

agents to make order out of this potential chaos [15]. The U.S. Department of Commerce reported that e-commerce sales grew 25.9% (in quarter Q12003 versus year ago) and has more than doubled in volume over the 2+ years of tracking. This is contrasted with a 4.4% increase in retail sales in the same period [5]. Clearly, e-commerce continues to expand its trade volume and overall importance to the world and national economy.

1.1 Complexity of a Typical E-Commerce Transaction

The development of electronic commerce is a logical progression that evolves with the technological progression of the Internet. The origins of electronic commerce find their roots with Electronic Data Interchange (EDI), where cooperating businesses (usually a business/vendor relationship) exchanged transactions (e.g., purchase orders) over private or Value-Added Networks (VAN) using exchange format standards such as ANSI X.12 and EDIFACT. These are referred to as business-to-business (B2B) relationships. As technology matured for the WWW, Web sites were created to serve static content, which were later improved to serve dynamic content from a database. Further development created technologies and techniques (e.g., multi-tiered systems) to support the transactional needs of e-commerce. The more common usage of the term e-commerce typically refers to business-to-consumer (B2C), in which a business is selling (fixed price or auction) to an individual over the Internet. The individual typically interacts with the e-commerce portal using a client machine and browser, while the portal is either hosted by the merchant or an application service provider (ASP). Electronic commerce sites are now common across the WWW and take many forms from retail merchants transacting online (e.g., Wal-Mart) to WWW-only merchants (e.g., Amazon).

A third type of marketplace, consumer-to-consumer (C2C) has also emerged, which is typically facilitated by a broker Web site (e.g., eBay).

E-Commerce transactions are inherently more complex than traditional retail transactions because they typically involve a complete workflow life cycle, as well as they are executed in a remote manner. For example, in a typical B2C transaction, based on an unmet need, the consumer would need to determine what to buy typically by searching virtual catalogs maintained by the e-commerce portals, compare and evaluate merchant alternatives to determine who to buy from, determine the terms of the transaction including price, purchase (usually through the traditional "shopping cart" function), arrange delivery and follow-up with the merchant on any product service issues. The burden is on the human user to execute and monitor each of these activities.

Beyond the workflow, several other factors contribute to the complexities of e-commerce applications. First, the e-commerce marketplace has been in a constant state of change. This change comes in the form of new functionality (e.g., online micro-payments) or new technologies (e.g., high-availability clustered solutions versus early solutions with single points-of-failure). Secondly, e-commerce solutions are inherently distributed and asynchronous requiring strict adherence to well-defined protocols and standards to maintain application integrity. Thirdly, transactions may also occur over a longer duration of time since the human user takes advantage of the virtual "shopping" experience where in traditional retail transactions, the "shopping" tends to be done in a shorter start to end timeframe (because of the physical nature of the shopping experience). This characteristic places an additional burden on the human user to continue to revisit e-commerce portals over the shopping duration to determine if any

decision criteria change over the timeframe. Lastly, the integrity requirements of an e-commerce system force the application to constantly monitor and deal with errors often in real-time (requiring an architectural infrastructure to support this capability).

1.2 Software Agents' Potential Role in E-Commerce Transaction Automation

The evolution of software agents has progressed from simple functions that assisted human users locally on their workstation (e.g., personal agents) to agents able to operate across a network (e.g., spiders). These advancements in combination with the explosive growth of the Internet have created opportunities for software agents to provide automation to the human user for e-commerce. In many ways, software agents closely resemble the human agent metaphor. The *Merriam-Webster Online Dictionary* defines agents as:

1. one that acts or exerts power;
2. something that produces or is capable of producing an effect;
3. a means or instrument by which a guiding intelligence achieves a result;
4. one who is authorized to act for or in the place of another.

The primary themes of autonomy, results-oriented, adaptive, power to act and the aspect of being intelligent are important characteristics of the human agent metaphor. Many of these characteristics are also true of software agents.

Although there is no universal definition, Franklin and Graesser [11] merge several attributes into a formalized definition that captures the essence of agents:

"An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future."

Franklin and Graesser's definition highlights a central premise, which is autonomy. As in the case of human agents, autonomy is achieved by delegating authority to the software agent to act on the users' behalf. With delegated authority, they must be able to act independently, communicate to their user, communicate to other agents or hosts, and monitor their state for effective decision-making. Their survey is cited often and their viewpoint on agent definition and classification, which stresses the proactive attribute, has helped shape the concept of agency and the difference between software agents and other types of programs.

Brenner et al. describe a simple taxonomy of "intelligent agents" that includes *transaction agents*, which have the ability to interact with e-commerce portals by monitoring and executing e-commerce transactions (the focus of this dissertation) [3]. They describe intelligent agents as programs that independently perform tasks on behalf of a human user in a network environment. By intelligence they also mean that the agent interacts with its environment by gathering information (e.g., knowledge of the user's wishes) and makes use of this knowledge by making decisions to satisfy those wishes in an autonomous manner. The terms agent, software agent and transaction agent will be used interchangeably in this dissertation.

In summary, although there remain many viewpoints on the discipline, most viewpoints commonly refer to the following formal attributes [6]:

- **Autonomous** – the agent has self control and is not dependent on the human user;
- **Reactive** – senses changes in its environment and responds in a timely fashion;
- **Goal-driven** – takes the initiative (i.e., proactive) to achieve the goal and has inferential capabilities within specific domains;
- **Persistent** – has temporal continuity of identity and state over long periods of time;
- **Social and Collaborative** – interacts and maintains a discourse with other software agents, hosts and/or users via an agent communication language (ACL);
- **Intelligent** – has the ability to adapt, learn and reason;
- **Mobile** – has the ability to transport itself from one host to another, while maintaining its state [27].

Software agents can play a significant value-add role in enhancing the e-commerce experience by improving the efficiency and quality of the interaction. Depending on the type of e-commerce (i.e., B2B, B2C or C2C), the role of the software agent and its ability to automate the e-commerce transaction is different. As mentioned earlier, in a B2C scenario, the human user is wholly responsible for executing and monitoring the entire workflow starting from needs identification to service and merchant evaluation. If a software agent were integrated into that workflow it could potentially:

1. Take the unmet need as defined by the human user as a set of product characteristics and/or constraints on product features to search and filter

unwanted products. This would result in a set of product choices that are acceptable to the user;

2. Likewise, use merchant-defined attributes such as price, warranty, service, product availability, and delivery time to determine which merchant to do business with;
3. Negotiate (if applicable) as in the case of an auction sale on the behalf of the user by again following pre-defined constraints and rules;
4. Settle the business transaction by making payment (typically credit card) and submitting contact details (such as shipping address) to complete the delivery portion of the transaction.

To-date, transaction agents have generally not been successful in automating the full life cycle of a typical e-commerce logical transaction. This life cycle begins with a human user identifying an unmet need, thus motivating the human user to satisfy that need by purchasing a good or service. As an e-transaction using a transaction agent, the life cycle continues with a discovery phase of appropriate electronic marketplaces followed by a search across those portals to locate the item of interest. As part of this search, the transaction agent would in theory evaluate the results of each portal with a logical reasoning capability to determine which opportunity best satisfies the unmet need and execute the purchase, payment and shipping steps on behalf of the human user. Today, this life cycle is only partially automated using transaction agents. In particular, the product search and merchant comparison phase are conducted by agents and the search results are typically presented to the human user for a decision. Where price is dynamic (e.g., auctions), price negotiation may also be automated using rules set by the

human user (e.g., auction bidding proxies). The purchase, payment and shipping events are typically not automated and executed by the human user. Although research continues, a transaction agent that is capable of acting on behalf of a human user to conduct e-commerce remains largely a research or prototype activity.

1.3 Serving Human Users and Software Agents: A Great Challenge

When Tim Berners-Lee designed the World Wide Web, he designed it for people to navigate/use, not software agents. The problem is that the World Wide Web was designed for serving static content and later evolved to support electronic commerce to human users and therefore, it is not conducive for software agents given the limitations of the Internet protocols (e.g., stateless HTTP) and front/back-end technologies and architecture.

Today's lack of a software agent interface for e-commerce portals is a major challenge for the success of software agents. To-date, these software agents have adapted and worked with the existing paradigm, which as mentioned was constructed with static content and human users in mind. So, for transaction agents to be truly successful in executing e-transactions, they need to be integrated in a more efficient and effective manner into the workflow of the e-commerce portal. Given the complex life cycle of e-transactions, this presents many challenges. These challenges include:

1. How might a transaction agent effectively communicate with its client-side host and e-commerce portals regarding its needs? What are the best approaches for the agents to communicate with its supporting environment? What triggers this interaction and transaction process?

2. How will the transaction agent describe its intent or objective to the e-commerce portal? What constraints may this communication involve? Will this communication require specialized agent communication languages?
3. Given the breadth of goods and services available across the WWW, how might an agent find the appropriate e-commerce providers and goods/services?
4. How might a transaction agent map a specific action (e.g., search the product catalog) to the proprietary business logic of each e-commerce portal?
5. Given the WWW-centric nature of this problem, how might the transactions traverse the Internet in such a way to not be blocked by the security infrastructure (e.g., firewalls) of the e-commerce portals? Also, how can the complexity (i.e., transport of function and temporal identity and state across e-commerce host) associated with agent mobility be minimized or eliminated?
6. Since e-transactions are dynamic in that the data involved with the transaction can change over short periods of time (e.g., goods availability, prices), how might an agent become aware of and react to these changes? How can conditions be set to not burden the client and transaction agents with trivial state changes in the parameters? How are these delayed transactions managed between the portal and agent?
7. How might results be presented to the transaction agent in order for it to evaluate the options using its logical reasoning capability?

1.4 Solution Strategy

To-date much of the research in this area focused on the agent. Multi-Agent Systems (MAS) operating within a standardized society of agents, agent communication languages, autonomous and mobile agent behavior models represent just a few of the strategies or techniques prevalent in the existing literature. The solution strategy described in this research provides an alternative viewpoint in which the focus is on the environment the agent operates within. This strategy distributes the solution across multiple components, which includes the agent, but also the client and server-side support systems.

The solution for this dissertation is based on a set of design principles that leverages proven techniques and technologies, thus enabling simpler design solutions. Design techniques such as a layered software architecture pattern, reusable components, proxies, and middleware are important aspects of the design solution and supporting systems for the transaction agent. The main design principles are:

1. Utilize existing Internet protocols, standards and technologies;
2. Support the mission-critical characteristics of an enterprise e-commerce portal for both the human user, as well as the software agent (e.g., availability, reliability, scalability, extensibility, performance, and flexibility);
3. Utilize open standards within the architectural solution to accommodate a broad range of e-commerce portals;
4. Create a solution that minimizes change: changes to the software agent, but more importantly, to the back-end business logic and database structure of the e-commerce portals.

The solution is first described in simple terms of an interoperability problem using Web services. In this scenario the e-commerce portal's business logic is exposed as an open application programming interface (API) using Web services, which has a number of advantages over other distributed computing techniques. This *Agent-Enabled Interface* (AEI) is further enhanced to allow the transaction agent to react to marketplace changes via delayed transactions, which originate from the server through the use of an additional server-side *Callback tier*. The client-side supporting system is also modified to work with transaction agents via a *Generic Middleware between Agents and Portals (GMAP)*. The feasibility of the solution is enhanced because the software architecture utilizes existing protocols and technologies and does not require new capabilities within the Internet. The solution is documented with the Unified Modeling Language (UML) wherever appropriate. Use case diagrams are used to illustrate the high-level requirements of the system, and in this dissertation, use cases and activity diagrams will be used in combination to illustrate examples of the proposed design solution satisfying the needs of typical e-commerce scenarios.

1.5 Major Contributions

This dissertation research describes design solutions that improve interoperability between transaction agents and e-commerce portals, and thus enhance their ability to execute back-end business transactions, while maintaining the traditional graphical user interface for the human user. This is a key enabler for solving the problem of the human user's inability to deal with the overwhelming products, options and choices available in today's e-commerce marketplace. Specific contributions include:

1. Designing an *Agent-Enabling Interface* that exposes an open business API that defines a robust interface for the software agent using cooperative client and server-side *Agent Interface Support Systems,* which enables the underlying infrastructure and back-end e-commerce architecture to be utilized without modification;
2. Extending this interface design with a server-side generic *Callback tier* to support the important software agent characteristic of reactivity thus ensuring that transaction agents can react to changes in the e-commerce environment in a timely and efficient fashion;
3. Defining a *Generic Middleware between Agents and Portals (GMAP)* that enhances interoperability between transaction agents and e-commerce portals;
4. Describing solutions that utilizes open standards and Web service technologies to support platform and implementation independent interoperability between agents and e-commerce portals.

The achievement of these capabilities can better enable transaction agents to automate the online transactions and leverage the marketplace opportunities that exist in the electronic world.

1.6 Dissertation Organization

The dissertation consists of six chapters organized in the following manner:

Chapter 1 – introduces the problem and complexities surrounding the e-commerce business model, as well as how transaction agents relate to e-commerce portals. It also defines the solution strategies and major contributions for this dissertation.

Chapter 2 – describes contemporary Web architecture, its limitation and highlights how e-commerce portals utilize this technology today. In addition, the classification of transaction agents (and their supporting systems) is introduced with some research/commercial agent examples. Lastly, a survey of the popular remote system integration techniques are described that play a role in the design solutions outlined in the dissertation.

Chapter 3 – describes a high-level solution for an Agent-Enabling Interface design that provides open access to the e-commerce portals business logic through a robust Web services API. The solution is extended into a generalized architecture that supports server callbacks to enable an efficient reactive transaction agent model. The software architecture can service both the needs of the human user, as well as a transaction software agent.

Chapter 4 – is an extension of the solution design described in chapter 3. It describes an architectural solution that enables the e-commerce portal to efficiently transact with pull/push transactional agents through the use of a server-side Callback tier.

Chapter 5 – describes a generic set of functions in a client-side support system including a Generic Middleware between Agents and Portals (GMAP), which supports interoperability among the transaction agent, service registry and business portals.

Chapter 6 – completes the dissertation with a set of conclusions, implications and recommendations for further research.

Chapter 2 – E-Commerce Portals, Transaction Agents and System Integration

This chapter explores the supporting literature that describes contemporary Web architecture, its limitations and how e-commerce portals utilize this technology today. In addition, the discussion of software agents will target a particular genre of agents referred to as transaction agents, which can be used to interact with e-commerce portals. The description includes typical functionality, integration methods and a description of various transaction agents and their limitations. Lastly, a survey of the popular remote system integration techniques will be presented that plays a crucial role in the design solution.

2.1 Contemporary Web Architecture and Its Limitations

E-Commerce has been enabled by the WWW and has emerged as one of the fastest growing elements of the Internet [14]. E-Commerce is commonly defined as the buying and selling of goods and services online. From the mid-1990's to-date, the most prevalent solution for conducting e-commerce over the Internet is through the World Wide Web, in which the human user is presented with a graphical user interface to interface with the e-commerce portal. Specifically, a contemporary view of e-commerce encompasses a broad range of issues including security, trust, reputation, law, payment mechanisms, advertising, ontologies, on-line catalogs, intermediaries, multimedia shopping experiences and back-office management [16]. E-Commerce portals generally have many users that execute a significant number of database queries (reading), with a

much smaller number of updates, but in total requiring a high performance and scalable solution.

Contemporary e-commerce portals use complex infrastructure to address mission-critical issues such as response time, page-serving capacity, fault-tolerance, automatic failover, security and disaster recovery. They also require advanced auditing and payment tracking capabilities for micro and macro online payments. These issues drive architectural solutions, which utilize redundant/clustered hardware, caching accelerators, load balancing, content delivery service providers (i.e., portal mirroring), distributed/replication database, and encryption/decryption accelerators. Many e-commerce portals have utilized the architectural lessons learned from the past and modeled the database to facilitate online transaction processing (OLTP) with an integrated data warehouse for data mining purposes.

In addition to a complex infrastructure, e-commerce portals typically utilize sophisticated and layered software architecture in combination with the infrastructure to achieve the functionality of the portal. Software architecture has progressed from a rather rudimentary use of 2-tier (not including the client), CGI programming solutions to n-tier, Java 2 Enterprise Edition™ (J2EE) or Active Server Pages (ASP with .NET emerging now) solutions. A software architecture based on tiers has the advantage of breaking a problem into smaller parts, making the solution easier because each part can solve a smaller sub-problem and each layer can be specialized. It is organized in a manner where each tier provides services to the adjacent tiers. A typical Web software architecture consists of four tiers: client, Web server, application server and database server. For this

reason, this is a common configuration for electronic commerce Web sites. The major design objectives of this architecture are:

- Clear separation of user interface, business logic and database access software layers;
- Easily scalable due to the separation of components and use of clustering;
- Secure architecture as the Web server is physically separated from the application server. All sensitive data is accessed from behind the wall of the application server tier;
- Higher level of security is possible by use of firewalls between layers.

Although most contemporary e-commerce portals are implemented in the traditional “tightly integrated” client/server architecture, a popular new alternative is emerging. Service-oriented architecture (SOA) is a client/server software design approach in which an application consists of software services and software service consumers. SOA differs from the more general client/server model in its emphasis on loose coupling between software components. Under the SOA model, applications are implemented by separating the interface aspects of the application from the core business logic. The business logic is organized into "services" each exposing a well-defined interface [24].

SOA is often described in terms of the different types of services it enables. In essence, a service is generally implemented as a discoverable software entity that exists as a single instance and interacts with applications and other services through a loosely coupled (often asynchronous), message-based communication model. Different types of service types include [4]:

- **Service** — a logical entity; the contract defined by one or more published interfaces;
- **Service provider** — the software entity that implements a service specification;
- **Service requestor** — the software entity that calls a service provider. Traditionally, this is termed a “client”; however, a service requestor can be an end-user application or another service;
- **Service locator** — a specific kind of service provider that acts as a registry and allows for the lookup of service provider interfaces and service locations;
- **Service broker** — a specific kind of service provider that can pass on service requests to one or more additional service providers.

In support of the higher-level abstractions of tiered and layered software are the protocols, standards and technologies inherent within the World Wide Web. These protocols, such as TCP/IP providing a flexible and reliable transport of packets across the Internet, together with other Internet-related standards and technologies, represent a foundation to support the communications and systems requirements of electronic commerce.

The World Wide Web consists of three main semantic components: Uniform Resource Identifiers (URI), Hypertext Transfer Protocol (HTTP) and Hypertext Markup Language (HTML). URI provides a universal naming convention for resources across the WWW. A popular form of a URI is a Uniform Resource Locator (URL). A URL is the address that defines the location of a file or resource on the Internet. The URL can consist of the protocol prefix, domain name, port number, subdirectory names and the file

name requested. It provides a mechanism to request methods that can respond with different responses. HTTP is a request-response protocol that enables Web components to communicate across the Internet. HTTP supports several request methods, with GET and POST being the most commonly used. A GET request (from a client) is typically used to retrieve information and is made to the resource specified in the URL with the generated response returned from the server. Unlike GET, POST is typically used to update information. A primary difference is in their technique for passing the data from the Web page form. GET encodes the information in the requested URL while POST puts the form data in the body of the request. There are limitations to the length of the URL string, so these differences matter in terms of the quantity of data passed as well as security considerations. Also, HTTP is a stateless protocol meaning the protocol does not maintain any information about the request once the response has been sent. This obviously complicates electronic commerce and software agents interoperating with these portals, so several mechanisms have been developed to maintain state where required. HTTP is often used to establish a connection between a client and Web server and transmit HTML files to the client's browser to render into graphical pages. The last semantic component, HTML is a standard for representing hypertext documents in ASCII format.

Today, a common way of accessing the WWW is through the use of a browser. The browser has evolved from a non-graphical simple program that allowed users to obtain resources from across the WWW to a sophisticated graphical interface that permits access to a variety of different Internet sources (e.g., email, news groups, chats, etc.). In the context of this research, the browser is a Web client, which constructs and sends an

HTTP request to a server and then receives, parses and displays the response. This dissertation utilizes the Java 2 Enterprise Edition™ (J2EE) specification for the design solutions. In that regard a small sub-set of the specification targeting the Web server tier will be utilized. Each of the tiers utilizes a container (e.g., applet, Web or EJB container), which manages the Java components. Java servlets and Java Server Pages (JSP) run on the Web server and are managed by the Web container. As typically the case, the design solutions utilize servlets to do session management while JSPs are used for presentation logic.

2.2 Transaction Agents and their Necessary Supporting Environments

Agents can be classified with taxonomies to better define and understand their capabilities. Researchers have developed several perspectives that highlight different attributes to better describe their features. These taxonomies include task-based classification (e.g., interface, personal information assistant, etc.), mode of behavior classification (e.g., mobile, autonomy, etc.), computational capabilities (e.g., finite state automata, stack automata, etc.), and AI paradigm classification (e.g., symbolic versus connectionist) [17].

Brenner's taxonomy of intelligent agents consists of human agents, hardware agents (e.g., robots) and software agents [3]. Software agents can be further decomposed into information, cooperation and *transaction agents*. Transaction agents, the main focus of this dissertation are generally thought of as a hybrid agent designed for conducting electronic commerce. These agents typically are designed for finding (i.e., information retrieval), brokering or negotiating, bidding or ordering and paying. This is consistent

with their view of intelligent software agents being programs that can perform specific tasks for a user and possessing a degree of intelligence that permits it to perform parts of its tasks autonomously and to interact with its environment in a useful manner [3]. To that end, transaction agents should be designed to prevent data inconsistency and integrity problems by satisfying the ACID properties of transactions (atomicity, consistency, isolation, durability). Atomicity ensures that either all or no steps of a transaction are executed. Consistency means that each transaction must maintain the integrity constraints on whatever object being modified. Isolation ensures that a transaction executes as if it were running by itself without interference from other concurrent transactions. Durability means that successful transaction changes are permanent, surviving any subsequent failure. These characteristics ensure that a transaction system remains consistent and predictable over time [32].

Maes et al. [22] utilize a framework that explores the roles of software agents as mediators in an e-commerce context (executing transactions). The framework is based on a Consumer Buying Behavior (CBB) model, which comprises the actions and decisions required in buying goods and services online. However, e-commerce covers a broad range of issues, some of which go beyond the CBB model (e.g., back-office management), but do not affect the usefulness of the model for describing the business process architecture. It is a powerful tool for understanding the roles of agents as mediators in electronic commerce [16]. CBB is defined as a six-stage model. The six stages are:

1. **Need identification** is the buyer becoming aware of some unmet need, which could be motivated by product information.

2. **Product brokering** is information retrieval, evaluation of alternatives based on user provided criteria resulting in a "consideration set." In essence, it is determining *what* to buy. Most e-commerce portals offer their products in the form of an "electronic catalog," which includes product descriptions and prices, which can be searched and evaluated.

3. **Merchant brokering** combines the "consideration set" with merchant information to determine *who* to buy from (includes an evaluation of merchant alternatives depending on price, availability, etc.).

4. **Negotiation** determines *how* to settle on the terms (price is usually not negotiable in retail but central to auctions). Negotiation deals with the additional dimension of time since retail versus auction have different durations and complexities.

5. **Purchase and delivery** is payment and selection of delivery options.

6. **Product service and evaluation** involves post-purchase services and customer satisfaction evaluations.

By cross-referencing the CBB model to commercial or research examples of transaction software agents, a functional gap is exposed, which is further exacerbated by the general lack of an industry accepted e-commerce interface for software agents (see Table 1). One obvious fact that is illustrated by this matrix is that software agents do not generally support functionality of need identification, payment/delivery or service/evaluation [22].

	Persona Logic	Firefly	Bargain Finder	Jango	Kasbah	Auction Bot	Tete@Tete
Need Identification							
Product Brokering	☑	☑					☑
Merchant Brokering			☑	☑			☑
Negotiation					☑	☑	☑
Payment / Delivery							
Service / Evaluation							

Table 1 – Online shopping framework with examples of agent mediation

The key conclusion of this framework is that today's software agents are capable of executing few of the actual electronic commerce transactions and do little where trust is required and monetary exchanges occur.

The evolution of software agents capable of transacting business on e-commerce portals has not progressed rapidly (as substantiated by Brenner et al. [3]). This is evidenced by the general lack of commercially available transaction agents on the World Wide Web today. There are a number of transaction software agents that exist in a research capacity that have added significant knowledge to the challenges and problems facing these agents [12].

The commercial agents available today generally fall into the category of information agents (e.g., spiders) or reactive/monitoring agents as in the case of the agent called "Eyes" (Amazon.com) that notifies you when books of interest become available [22]. Also, some auction sites (e.g., eBay, Amazon) have a local proxy "agent" that will bid on the users' behalf up to a maximum amount set by the user. In general, transaction

agents were developed using the public interface of the e-commerce portal to execute transactions in a narrowly defined view of e-commerce. For example, BargainFinder and Jango are examples of transaction software agents that utilize a proprietary "wrapper" to scrape Web pages for product and merchant content [16]. Some examples of transaction software agents are:

- BargainFinder [20] was a proof-of-concept agent developed by Andersen Consulting and was the first agent that could do online price comparisons across e-commerce portal (working through the public interface using typical HTTP requests). Merchant brokering was restricted to retailers who subscribed to the service, but some of the retailers block price check queries, which limited its effectiveness. Jango [22] was similar, but more closely approached an autonomous agent by originating from the user's Web browser (instead of centrally as in the case of BargainFinder) and thus appeared to the e-commerce portal as a "real" customer.

- MIT Media Lab's Kasbah [21] was architected as an online marketplace (Internet host), multi-agent consumer-to-consumer system for the buying and selling of goods. A user could set the agent's behavior criteria and dispatch it into the marketplace. The criteria include price, time constraints and quantity of merchandise desired. Users could configure the agent with three different negotiation strategies: anxious, cool-headed and frugal. It utilized a simple linear, quadratic or exponential function for increasing its bid over time. The Kasbah seller agents proactively sought out potential Kasbah buyer agents. The system was also designed to have a trust and reputation function called

the "better business bureau," in which upon completion of a transaction, both parties could rate how the other party performed (somewhat like the eBay rating system for the human buyers and sellers).

In addition to these research/prototype examples, work is being done on standard open agent architectures that are robust in nature. To achieve the attributes required of an intelligent software agent (i.e., transaction agent), the agent will need to effectively interoperate with the e-commerce portal. In order to accomplish this, agent enabled standards are critical. Initiatives undertaken by FIPA (Foundation for Intelligent Physical Agents) and OMG (Object Management Group) attempt to bring standardization to the research. Several standards have emerged with the most successful being the FIPA Agent Specification [9][28], OMG Mobile Agent System Interoperability Facilities Specification (MASIF) [30] and the Knowledge Query and Manipulation Language (KQML) [7].

FIPA is a collection of international commercial companies whose main objective is the development of specifications for open agent interfaces. These specifications were originally developed in 1995 and have been organized into four areas that are published on the organization's Web site [10]. These areas include agent communications, management, agent/software integration and human/agent integration. Agent communication specifies how communications occur between agents through the use of an agent communication language (ACL). The communications include negotiation, cooperation and information exchange. Agent management focuses on the facilities required to support the location and creation of agents, as well as communications, security and mobility. Agent/software integration is the interface between agent and non-

agent software, while human/agent integration targets the interface and interaction between the human user and agent. These specifications do not prescribe the internal architecture of agents, but they do describe the interfaces necessary to support interoperability between agent systems [28].

Also, research has been done on agent communication languages and languages such as Knowledge Query and Manipulation Language (KQML) [7] and FIPA's Knowledge Interchange Format (KIF) [36], which have been developed for inter-agent communication. KIF is a form of first order logic for encoding the content of a knowledge base. KQML was developed by the ARPA (now DARPA) Knowledge Sharing Effort (KSE) to facilitate sharing and reuse of knowledge bases [7], and is widely used for software agent development. Its primary focus is on pragmatics (i.e., who and how to communicate), while secondarily concerned with semantics. KQML is a message format and protocol that supports run-time messaging between agents and hosts. The construct of KQML messages consist of a performative, which defines the permissible "speech acts," and associated arguments. Since agents can act in an autonomous and asynchronous manner, they may have different or conflicting agendas, so the meaning of KQML messages are controlled and constrained by the sender. The receiver can choose an appropriate course of action from its viewpoint, thus allowing it to maintain its autonomy (to maximize the relationship, it is preferable to have a cooperative relationship, but this is not always possible) [1]. Finin et al. describe several levels that agent-based systems must agree upon to interoperate [8]:

1. **Transport** – how agents send/receive messages;
2. **Language** – what the messages mean;

3. **Policy** – how agents structure conversations;
4. **Architecture** – how to connect systems according to constituent protocols.

The language can be described to have three layers: content, message and communication [1]. The content layer contains the message in the application's representation protocol (e.g., ASCII). The message layer defines the interactions possible with the KQML speaking agent. It includes the message protocol, performative, as well as the message type, language and ontology information. The communication layer manages the information pertaining to the identity of sender/receiver, as well as a unique identifier for the messages.

Another standard, MASIF (Mobile Agent System Interoperability Facility) [30], is for mobile agent systems, which was developed by OMG in 1998. MASIF addresses interoperability between agent systems (i.e., a platform that can create, interpret, execute, transfer and terminate agents – a host can contain multiple agent systems). In essence, it enables interoperability between agent platforms of different vendors. The MASIF reference model describes a collection of definitions and interfaces that provide an interoperable interface for mobile agent systems. The agent system type describes the profile of an agent, meaning the vendor, language and serialization mechanism. Agents are transported between places (where the agent can execute), which are associated with locations that specify the place name and address where the agent system resides. An agent system can contain one or more places and a place can host one or more agents. When a client requests the location of the agent, it receives the address of the place where the agent is executing.

The reference model defines two interfaces: MAFAgentSystem (for agent transfer and tracking) and MAFFinder (for registration and maintenance of a dynamic name and location database of agents and agent systems). MASIF standardizes:

- **Agent Management** - an administration function managing the creation, suspension, resumption and termination of agents;
- **Agent Transfer** - mechanisms for receiving agents and retrieving their classes, which requires cooperation between different agent systems that is complex to accomplish;
- **Agent and Agent System Names** - standardized syntax and semantics of agent and agent system names allow agent systems and agents to identify each other, as well as clients to identify agents and agent systems;
- **Agent System Type and Location Syntax** - the agent transfer is limited to agent system implementations that are compatible (i.e., agent system type can support the agent). The location syntax is standardized so that the agent systems can locate each other.

Given its OMG heritage, MASIF was designed to leverage CORBA services such as Naming, Lifecycle, Externalization and Security. For example, both MASIF interfaces are defined as CORBA objects allowing them to be published in the Naming Service. Agents and agent systems aren’t required to be CORBA objects, but if they are, they can utilize these services.

2.3 Remote System Integration across Heterogeneous Platforms

Heterogeneous system solutions to solve business problems have been the norm and continue to grow with the emergence of new business models such as e-commerce. Several interrelated trends are driving the design and increasing need for remote system integration across heterogeneous platforms [23]. The rapid adoption of the networked computing model via the Internet, further enabled with broadband data capacity is the single most significant change motivating the need for remote system integration. Also, the increasing complexity of systems and software requires a "divide-and-conquer" strategy to make the problem and solution more manageable, and in combination with the networked computing model, drives the need for remote system integration. Lastly, emerging computing models such as application service providers (ASP) in which applications are provided on a rental basis depend on strong systems integration. ASPs can host simple systems (such as email and Web servers) or more complex e-commerce applications in which B2B and remote system integration are important elements of the hosted solution [34].

The foundation for the communication between the distributed parts of an application is the remote procedure call (RPC). Developed in the 1980's, it allows a procedural program to call a function that resides on another computer as conveniently as if that function were part of the same program running on the same computer. It furthers the goal of divide-and-conquer by allowing developers to concentrate on the application without concern for whether the function calls are local or remote or other network details (e.g., it performs all the networking and marshaling of data). To simplify the use of this technology, middleware was developed that sits between the server's operating

system/networking services and business applications. The most successful middleware solutions are CORBA, Java RMI, .NET and Web services. The challenge for middleware pertains to efficiency and effectiveness across the heterogeneous landscape. Each of the middleware specifications will be described below highlighting some of their advantages and disadvantages.

2.3.1 Common Object Request Broker Architecture (CORBA)

The Object Management Group (OMG) [29] developed the specification for CORBA and the related Internet InterORB Protocol (IIOP), which is the communication protocol between Object Request Brokers (ORB). The CORBA architecture, referred to as Object Management Architecture (OMA), is a set of interfaces for CORBA applications. The Object Request Broker is at the core of CORBA providing basic object connectivity, while CORBAservices and CORBAfacilities provide value-added higher-level services to CORBA objects and CORBA applications respectively. The ORBs are middleware objects that allow requests from the client objects to be delivered to the server objects and the responses to be delivered back [18]. CORBA can be used to wrap a legacy application to expose CORBA interfaces for remote calls from clients across a network. The CORBA interfaces of server objects are defined at a high level of abstraction using the Interface Definition Language (IDL). IDL is language independent and it maps to the major programming languages for compatibility. This interface serves as a contract that the server offers the clients that want to invoke it. Based on an IDL interface, a stub object will be generated and deployed on the client-side, and a skeleton object will be generated and deployed on the server-side. The stub object marshals the request parameters and passes the invocation to the remote skeleton object through

ORBs, and the skeleton object un-marshals the request parameters and invokes its local server method implementation. This same process works in reverse for the response.

The IIOP is the common protocol that ensures compatibility of operation and parameters. Although the stub on the client-side and skeleton on the server-side may have been compiled into different languages, the common IDL interface definition ensures compatibility. These compiled objects may also run on different ORBs from different vendors. Each CORBA object instance has a unique object reference, which is critical for supporting remote invocation. To invoke a remote object instance, the client may obtain the CORBA object reference through a naming service (part of CORBAservices) and make the same local call but using the object reference of the remote service. The local ORB recognizes that the object reference is remote and thus routes the request across the network to the remote ORB. This separation of interface from implementation is the essence of CORBA and how it maintains transparency of its interoperability [29].

The main advantage of CORBA is that clients and servers can be written in any language and run on any platform, thus enhancing openness and interoperability that can address the heterogeneous characteristics of the problem defined in this dissertation. Also, it is a mature technology, so it has been proven to be a reliable technology that scales well.

The main disadvantage of CORBA is complexity. The services defined in the architecture are difficult to use and few vendors have implemented CORBA services in their products. Also, since the ORB usually presents the service on a port other than port 80, special packet tunneling in the intervening firewalls may be necessary to ensure the

method invocation can get through. Also, solution deployment is more difficult with CORBA because the client must have a CORBA ORB installed, which has financial implications and requires careful version control.

2.3.2 Java Remote Method Invocation

The Java Remote Method Invocation framework is Sun's technology to enable Java objects distributed on different Java virtual machines (JVM) to communicate via TCP/IP with normal method calls. Like CORBA, the local object can make a call to a remote object by addressing it with its object reference. In the case of Java RMI, this is achieved by the client either querying the server's RMI registry (RMI doesn't support services such as provided by CORBAservices) or by using a reference passed as an argument or return value. RMI uses the serialization class to marshal and un-marshal parameters between the client and server objects. Since only Java objects can be on each side of the connection, an IDL is not required to define the objects in a high-level abstraction [18].

The development of an RMI interface is a straightforward server activity. The remote interface is defined/designed (meaning identifying which methods are available to call). A subclass is created from the appropriate RMI server class and compiled to generate the stub and skeleton files. The remote object interface is then registered with the RMI registry.

The main advantage of RMI is that since the scope of use is limited to Java, the use of the middleware is less complex. RMI allows programmers to avoid the complexities of communication protocols between applications, as well as the use of streams and sockets. However, since only Java objects can be supported, it is a major

disadvantage in heterogeneous solutions. Also, since RMI is built on top of sockets (and uses serialization), it will be slower than a socket call, but competitive with the other RPC techniques. As in the case of CORBA, the RMI service (registry and method call) will be on a port other than port 80, thus packet tunneling may be needed in the intervening firewalls to ensure the method invocations can get through.

2.3.3 Microsoft .NET

.NET is the evolution of Microsoft's distributed computing technology. It is a comprehensive family of Microsoft software products that are built upon industry and Internet standards (e.g., Web services). It has three major parts including the .NET Framework, Visual Studio .NET (an integrated development environment for several programming languages) and .NET Enterprise Servers. The .NET Framework has the base technology including the Common Language Run-time and unified class library, which contains ASP.NET. ASP.NET provides a low-level programming model equivalent to ISAPI (an API extension to Microsoft's Internet Information Server and other Web servers that enables Web-based applications to run much faster than conventional CGI programs). ASP.NET Web services also supports service request using SOAP over HTTP, as well as normal HTTP request-response (GET/POST) operations. Like other Web services implementations, ASP.NET Web services does not expose the server-side data types to the client and operate in a stateless manner [26].

To support the RPC model, Microsoft implements .NET Remoting in the architecture. .NET Remoting enables method calls across domains and machines, including remote network calls. For efficiency of more localized calls, it supports a binary protocol over TCP/IP and the SOAP protocol over HTTP/SMTP. It enables

tightly coupled integration between client and server and clients can call server-side objects through unique references. In addition, clients can control the lifetime of these server-side objects essentially implementing a stateful relationship (the client-side applications also must be developed with .NET Remoting to enable this feature) [26].

2.3.4 Web Services

Web services are a relatively new set of standards and technologies that have evolved from component and distributed computing architectures to satisfy the need of integrating applications written in many different languages with many different data formats. As the WWW is oriented to visual traversal or sometimes referred to as eyeball Web for program-to-user interaction, Web services targets the transactional Web for program-to-program interaction [13]. It represents the convergence of the service-oriented architecture and World Wide Web. Since Web services are based on the same paradigm as generalized distributed service-oriented architecture, it maintains the same four components of service broker, provider, requester and locator [24]. The client/server architecture has continuously improved to support high performance, stable and reliable systems and Web sites through the use of tightly coupled distributed computing protocols such as RMI. By contrast, the objective of Web services is to create interoperability through loosely coupled connections that are vendor, platform and language independent [26]. Also, unlike CORBA, .NET and RMI, Web services does not have the intelligence for understanding how to map the message into an RPC in the interface itself, but rather as part of the XML processor, which processes the message and follows the associated instructions (i.e., WSDL) for parsing and mapping it into the service it defines.

Web services are not accessed via object-model specific protocols, such as the competing solutions (i.e., RMI, IIOP, DCOM), but rather use the widely accepted Web protocols. As compared to other distributed computing techniques, Web services provides a layer of abstraction above CORBA or .NET servers and instead of middleware, is better thought of as a message queuing system. Web services are still evolving, but they can be described in the following manner:

"A Web service is an interface that describes a collection of operations that are network-accessible through standardized XML messaging. A Web service performs a specific task or a set of tasks. A Web service is described using a standard, a formal XML notation, called its service description that provides all of the details necessary to interact with the service, including message formats (that detail the operations), transport protocols, and location. Web service descriptions are expressed in WSDL." [13]

As defined, Web services are based on several de-facto standards that are required to build and deploy them. These standards provide for data definition, service description, message transport and service discovery. They are [25]:

- **Extensible Markup Language (XML)** - XML is a standard set of rules for defining semantic tags that break a document into parts and identify the different parts of the document. Unlike HTML, which has a set number of tags and describes the format of the document, XML allows you to create the tags required to properly identify the semantics or structure of the document (hence its extensibility). It is essentially a meta-markup language or a language for defining other languages. XML allows designers to develop any arbitrary set of tags to describe meaning and hierarchical structure of data and supports specification of sophisticated data types required for efficient data interchange between different programs and systems. This meaning and

hierarchy is described in a simple, flexible, human-readable text format using either a Document Type Definition (DTD) or an XML Schema.

- **Web Services Description Language (WSDL)** – is defined in XML and it describes the services exposed and the specifics of what service requesters must adhere to in sending SOAP messages (it also corresponds to the CORBA IDL). The parts of a WSDL document are (although developers generally do not need to understand these details):
    - *definitions* is the root of the WSDL document and contains the other parts and namespaces the WSDL document can use;
    - *message* is defined for both the request and response (the XML format is specified by linking to the definitions in an XSD schema) and represents types of variables that service providers and requestors pass between each other;
    - *operation* lists the messages contained in one request-response (input-output) message flow and maps the messages in the *message* element to the actual service (method parameter or method return type);
    - *portType*, which is an abstract endpoint describing a single request-response operation in the message element that defines how the input and output messages associate with the operation;
    - *binding*, which is a concrete endpoint for the portType specifying how the service provider and requestor should send messages (in various protocols SOAP, MIME, HTTP) between them;

  - *service*, which defines the endpoint (a particular URL address to the e-commerce portal).

- **Simple Object Access Protocol (SOAP)** – is the way a program running in one type of operating system can communicate with a program in the same or different operating system by utilizing HTTP and XML. It is a XML-based messaging protocol used to encode the information (in this dissertation it encodes the remote method call);
- **Universal Description, Discovery and Integration (UDDI)** – is an XML-based directory for Web sites that allow them to advertise themselves over the Internet (similar to the CORBA Trading Service). These registries are document repositories containing the WSDL documents representing the Web services. UDDI is organized into three main categories that can be queried. They are: white pages for business name, address and contacts, yellow pages for type of business, location, products and green pages for technical information about the services such as how to interact with them. The registration data is comprised of five major data structures: businessEntity, businessServices, bindingTemplate, tModel and publisherAssertion, which are assigned universally unique identifiers (UUID) for query support. The first three parts are critical since they describe the business, services and information about the services needed for accessing them. UDDI supports two SOAP-based APIs for the data model; one is used during the registration process to publish and maintain the information and the other is used to query the registry [35]. In addition, Java WSDP supports an API for XML

Registries (JAXR). JAXR enables developers to use a single, easy-to-use abstraction API to access a variety of XML registries. A JAXR information model describes content and metadata within XML registries [19].

A key advantage of Web services is that it can be envisioned as a wrapper that defines an interface for interacting with the back-end of a system. The interaction can be either RPC-oriented or document-oriented. Web services support a request-response paradigm typical of a synchronous RPC-oriented communication, but through emulation where the XML processor rather than the protocol itself correlates requests with replies (which is quite different from CORBA). The Web services emulation of RPC can be mapped to traditional RPC-based systems such as CORBA, EJB or DCOM. In the RPC-oriented scenario, a SOAP encapsulated XML message that conforms to the service as defined in the WSDL is received by the server via HTTP, where it is parsed and transformed into the appropriate method call that the back-end software understands. The response follows the same transformations in reverse back to the client. RPC-oriented interactions are typically modeled as synchronous processes although an asynchronous interaction is also supported (via Java API for XML Messaging - JAXM).

JAX-RPC (Java API for XML-based Remote Procedure Calls) is a synchronous mechanism that allows programmers to create and access Web services by using XML-based remote procedure calls. It provides a simple, RPC-oriented API that hides the underlying details of the SOAP communications and WSDL descriptions. The architecture is layered with the server-side containing a JAX-RPC service run-time environment and service end point while the client-side runs a JAX-RPC service run-time

environment and client application. An example of a typical RPC-oriented SOAP message request and response for an order status check is as follows [26].

```
Request-
        <SOAP-ENV:Envelope
        . . .
                <SOAP-ENV:Body>
                        <m:CheckOrderStatus
                        xmlns:m="www.xmlbus.com/OrderEntry"?>
                                <orderno>12345</orderno>
                        </m:CheckOrderStatus>
                <SOAP-ENV:Body>

        </SOAP-ENV:Envelope>

Response-
        <SOAP-ENV:Envelope
        . . .
                <SOAP-ENV:Body>
                        <m:CheckOrderStatusResponse
                        xmlns:m="www.xmlbus.com/OrderEntry"?>
                                <status>shipped June 18</status>
                        </m:CheckOrderStatusResponse>
                <SOAP-ENV:Body>

        </SOAP-ENV:Envelope>
```

In this example, the request sends the order number as a parameter to the order status service, and the response message returns the most recent status. A CheckOrderStatus request is sent to the OrderEntry service. The JAX-RPC run-time workflow is as follows:

1. The client calls a remote method by invoking a method of the same name on a stub, a local object that represents the remote service;
2. The stub invokes routines in the client-side JAX-RPC run-time system;
3. The run-time system converts the remote method call into a SOAP message and then transmits the message as an HTTP request;

4. When the server receives the HTTP request, the server-side JAX-RPC run-time system extracts the SOAP message from the request and translates it into a method call;
5. The JAX-RPC run-time system invokes the method on the tie object;
6. The tie object invokes the method on the implementation of the OrderEntry service;
7. The run-time system on the server converts the method's response into a SOAP message and then transmits the message back to the client as an HTTP response;
8. On the client, the JAX-RPC run-time system extracts the SOAP message from the HTTP response and then translates it into a method response for the client application program.

JAX-RPC has significant advantages over RMI and CORBA. It enables applications to utilize SOAP and WSDL to invoke Web services on non-Java platforms (and vice versa). Although RMI/IIOP does enable this interoperability, JAX-RPC is much easier to use (much easier than CORBA as well). Lastly, since the SOAP message is transmitted via HTTP to port 80, there is no concern about additional firewall configurations that would be required to allow this traffic to pass through the firewall.

Chapter 3 – Solution Strategies

This chapter describes the solution strategies that target the interface related problems of software agents interoperating with e-commerce portals. The solution is first scoped as an interoperability problem using Web services. In this scenario typical supporting systems are required on the client and server to expose the business logic of the e-commerce portal. This Agent-Enabling Interface (AEI) design is further enhanced to allow the transaction agent to react to marketplace changes via callback from the server, which represents a new portal software architecture that utilizes a Callback tier. The client-side support system is also modified to work with transaction agents. This modification represents an evolution of the client-side supporting system into a Generic Middleware between Agents and Portals (GMAP). These architectural enhancements in conjunction work to achieve a specific set of principles and objectives, which are the basis of this dissertation. The chapter begins with a review of the main principles, objectives and assumptions that articulate the basic premise and claims of the dissertation and shape the design solutions described in the document. A description of contemporary architecture for electronic commerce is described as a foundation to the problem statement and dissertation idea.

3.1 Problem Statement

Current e-commerce portals are designed to support efficient interactions with human users through Web browsers. Humans need to manually search for interested portal Web sites, and visually scan the information among rich Web page components and text.

With the ever-increasing number of e-commerce portals, a human user is faced with a large number of choices for similar services or products, and these choices are usually coded in quite different visual presentations. This will worsen as the growth of the Internet and e-commerce continues unabated. To partially alleviate the burden of such a decision making process, researchers are proposing to use *transaction agents* to automatically or semi-automatically search for potential e-commerce portals, compile information of interest, and execute best choices for human users to consider. But based on today's portal architecture, the agents can only simulate humans to search through Web components and text for related information, which is inefficient and inaccurate. Furthermore, since the traditional Web architecture is based on a client/server model, the portals have no means to notify the agents, when they operate on client platforms, of any change of information or the advent of events interesting to users. Such two-way communications between agents and portals could open doors to more advanced transaction agent services.

The underlying problem is that the World Wide Web was designed for serving static content to human users. Today's lack of a software agent interface for e-commerce portals is an inhibitor for the success of software agents. To-date, these software agents have adapted and worked with the existing paradigm (graphical and form-based HTML interface), which was constructed with static content and human users in mind. For software agents to be truly successful in the electronic commerce world, they need to operate in a more efficient and effective manner through a new architecture and agent interface.

3.1.1 Principles

Several design principles were adhered to that enable a practical solution to be devised, which could be implemented with typical portals found in today's e-commerce marketplace. *First, the solution outlined in the dissertation uses existing Internet protocols, standards and technologies.* Although improvements in some technologies (e.g., protocol state maintenance, semantic searching, etc.) could enhance the effectiveness and ultimate success of transaction agents, the principle of design to the current state-of-the-art was utilized. *Second, the solution must support the mission-critical characteristics of an enterprise e-commerce portal for both the human user and the software agent (e.g., availability, reliability, scalability, extensibility, performance, and flexibility).* The e-commerce portal should provide the same level of support and service to both the human user and software agent. *Third, utilize open standards within the architectural solution to accommodate a broad range of e-commerce portals.* In order to more easily interoperate with a variety of e-commerce portals, XML and other appropriate Web services technologies can enhance the interaction between software agent and portal. *Lastly, to create a solution that minimizes change: change to software agent design, but more importantly, the presentation and business logic and database structure of the e-commerce portals.* The solution should address the interface requirements of the human user and software agent while keeping the portal's back-end application/business logic and database unmodified.

3.1.2 Objectives

As stated, the overall goal is to facilitate transaction agent and e-commerce portal interoperability via exposing and reusing the powerful and robust source of business logic encapsulated within the portal. To accomplish this goal, the following objectives must be achieved:

1. Providing efficient e-commerce portal interfaces for both browser-based human clients and client-side software applications including transaction agents or their supporting 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. Adopting component-based system design and maximizing reuse through generic functions;
5. Avoiding a custom software installation for an agent to interact with a particular e-commerce portal through use of publicly available service definitions.

3.1.3 Assumptions

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

- That the interaction is between the transactional agent and the client-side support system where they relate to each other and the outside world as black boxes, meaning that they only expose an API and no internal design or data

assets. Inter-agent interactions (i.e., multi-agent systems) are not dealt with in the architecture;

- That the design solutions are primarily oriented at solving a B2C retail problem. Transaction agents targeted at B2B or C2C e-commerce are out-of-scope of the solution (although many of the design concepts apply);
- The solution is based on the Java 2 Enterprise Edition™ Platform Specification although the concepts can apply to other application frameworks as well;
- Security, although critically important is assumed to be out-of-scope of the architectural solution;
- The e-commerce portal is designed with contemporary tiered architectural concepts that separate presentation, business and data management logic;
- A performance comparison of the design solutions defined in this dissertation compared with existing methods is assumed to be out-of-scope.

Although transaction agents may have many characteristics and features required for achieving the full breadth of functionality, this dissertation focuses on the interface to/from the agent and its interaction with the outside world via the supporting system on the client.

For purposes of this research, a transaction agent is defined as a standalone application running on the client and interacting with a client-side support system. The client-side support system interacts with both the server-side support system and the transaction agent. Several assumptions are made regarding the design capabilities of the transaction agent:

- They are *generic* in nature, meaning they are designed to work with a broad range of portals and are not specifically designed for a particular portal;
- The agent can query tables or in-memory objects for input criteria and results returned (i.e., feedback from previous method calls) from the server-side support system and requires no other sources of information;
- The agent can parse the human user input statement or browser-based form and convert the business transaction specification (i.e., the purpose of the human user) into individual attributes;
- It possesses an internal logical reasoning capability for evaluating the input criteria (i.e., the business transaction that the human user wants to accomplish) and results returned from the portals for decision making (the details are outside the scope of this dissertation);
- Using the input criteria and internal logical reasoning capability, the transaction agent can generate the sub-transactions (i.e., the individual instructions or steps required to achieve the business transaction) and create additional sub-transactions from results from new inputs or previous method calls (i.e., feedback);
- It is expected that the transaction agent provides support for primary use cases scenarios where secondary scenarios that involve error conditions may require human user interaction;
- The agent can multitask and process multiple business transaction and result streams concurrently;

- The transaction agent operates in a manner consistent with the Consumer Buyer Behavior (CBB) workflow thus enabling support for a full range of e-commerce transactions.

As mentioned, although significant progress has been made, to-date, transaction agents have not generally been successful in automating the full life cycle of a typical e-commerce logical transaction. These assumptions pertain to a transaction agent design that would be important to achieve full life cycle support. All other capabilities needed to achieve the objectives set out in this dissertation (e.g., life cycle management, scheduling, etc.) will be handled by the supporting systems on the client and server-side tiers.

3.2 Semantics between Generic Transaction Agents and Portals

A significant challenge for generic transaction agents in interacting with e-commerce portals is how to interpret and understand the method call semantics used across a wide range of portals. There are few standards that constrain method names and signatures so the variety found throughout the industry would be a problem for generic agents. The lack of standards or formal approaches to the problem in this area is a gap that may be a contributing factor to the slow evolution of transaction agents for e-commerce.

To address this problem, several alternatives are suggested in this dissertation. Some of these alternatives have a basis in the research to-date as transaction agents have evolved over the years. Some options are:

1. Lessen the "generic" characteristics of the transaction agent and custom develop their capabilities to work with individual portals. This approach has some basis in the industry in the form of agents that employ "screen-scraping"

to interact with the portal (a popular technique used with "spiders" – a form of mobile software agent). In addition to the burden of developing a variety of these custom agent interface wrappers, there are a few other disadvantages with this approach. First, the human user would need to interact with a large number of different agents to closely monitor and coordinate their activities since achieving the business transaction will likely involve multiple portals. This means that the human user is doing more of the work and there is less automation of the e-commerce workflow. Also, since the custom agents deployed across multiple portals would not operate in a coordinated manner, the human user would need to manage the decision making process to ensure the "best" option was selected. BargainFinder [20] and Jango [22] were examples of agents that used this proprietary interface "wrapper" to scrape Web pages.

2. Create a common marketplace with a dedicated transaction agent that is designed to interact with this marketplace. In this scenario the merchants (i.e., e-commerce portals) would post information about their goods and services in this common marketplace that the dedicated agent would interact with. In essence, this approach moves the goods and services to the transaction agent versus the agent visiting the marketplaces. The obvious disadvantage of this approach is that the e-commerce portals have competitive reasons to maintain their own separate marketplace and would have little incentive to co-aggregate their goods and services with other merchants. Kasbah [21] is an example of

an agent that utilized this approach of exploiting a common marketplace (referred to as the “Kasbah marketplace”).

To address this problem, we propose a strategy of developing a semantic understanding between a generic transaction agent and portal that enables the agent to invoke the correct business logic method, and properly interpret the parameter and return values. The methodology for creating semantic meaning between the intent of the transaction agent (and thus human user) and the specific method call of the e-commerce portal is through the use of *action words*. By codifying the method name, parameter and return value names to action words, the e-commerce portal and generic agent can interact with a semantic basis of understanding. If standardized, these action words could greatly simplify the semantics between the agent and portal (the CBB model could be used to provide some foundation for consistency and standardization of these action words). The action words representing the method name can be standardized, while more flexibility is required to semantically represent the parameter and return values, so although not standardized, the action words for these attributes can be utilized as key words by the agent’s logical reasoning capabilities. A proposed partial list follows:

Action Word	Description
MethodName-	
SearchProductCatalog	Search for a key word or product identifier
CheckInv	Check availability of inventory
CheckPrice	Check price of item of interest
Browse	Browse and return descriptive information about item of interest
CreateOrder	Create order and quantity
UpdateOrder	Update order and quantity
ProcessOrder	Process order and quantity (make payment)
CheckOrderStatus	Check order status
Parameter or Return Values-	
ItemID	The identifier for the item on the portal (e.g., ISBN)
ItemPrice	The price of the item

CurCode	Currency code for pricing
PromoCode	A special discount pricing code (coupon)
…	

Table 2 – Action Words

It is recognized that this knowledge representation challenge is a difficult problem that without an industry accepted standard makes the operation of a generic transaction agent difficult. To that end, significant research is underway in the area of the Semantic Web where the use of XML, Resource Description Framework (RDF), and domain-specific ontologies are powerful tools for developing a structured collection of information and inference rules that can assist automated reasoning (e.g., transaction agents) [2]. The solution described in this research can be seen as a basic framework that narrowly targets the challenge of transaction agents and can be enhanced to span the entire e-commerce workflow across different types of portals.

3.3 Contemporary E-Commerce Portal Design

This section describes the design of a typical e-commerce portal, which will be evolved to include more detail in later sections.

3.3.1 E-Commerce Use Cases

The context of the use case for an e-commerce portal involves the human user as the main actor and the e-commerce portal. For e-commerce portals there are typically external actors such as payment services (for online credit card charges), currency converters, shipping systems and such. Other external actors that are internal to the e-commerce environment might be data warehouses, inventory and back-end accounting systems (see Figure 1).

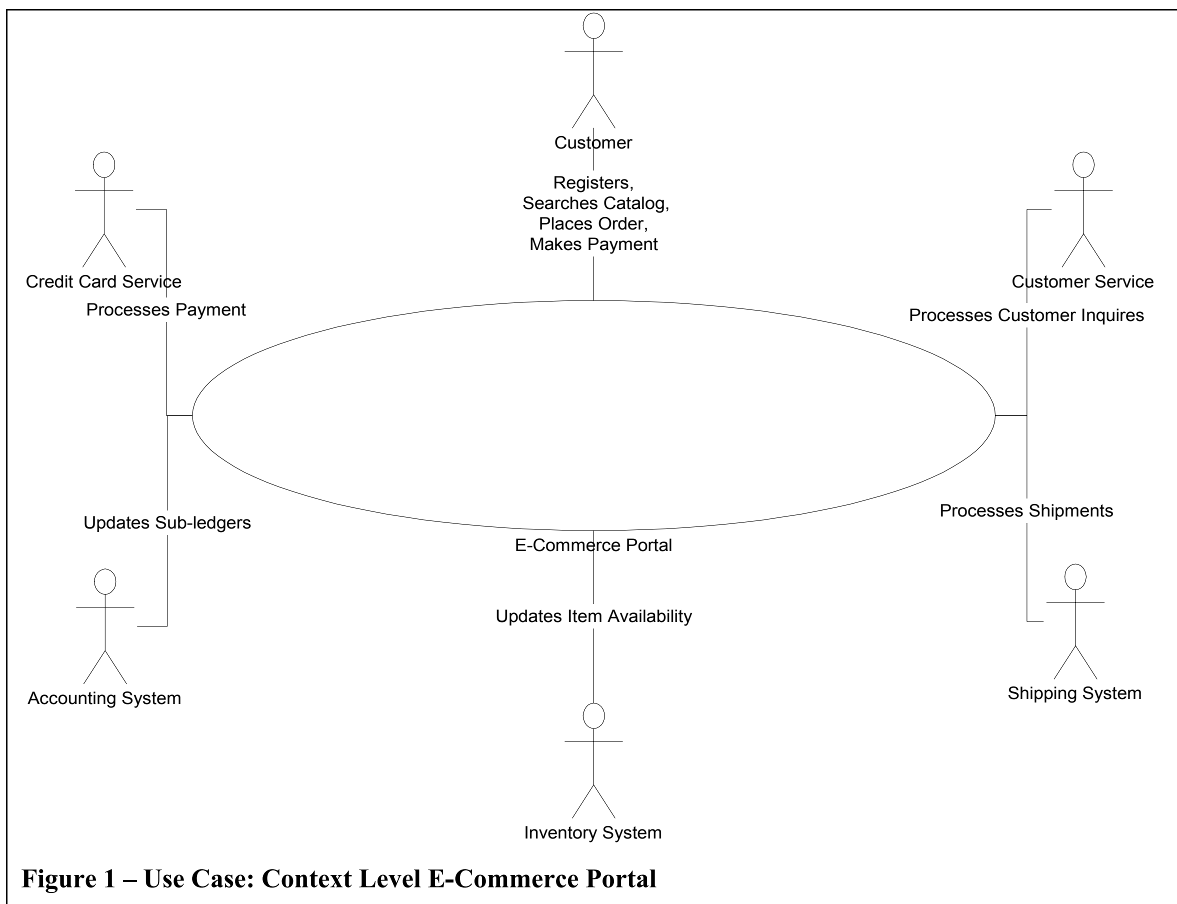

Figure 1 – Use Case: Context Level E-Commerce Portal

In the e-commerce scenarios, the human user is motivated to purchase an item to satisfy some unmet need and therefore visits various e-commerce portals he/she are aware of from experience, advertising or other communication channels. The user conducts a search across these portals for the item of interest with several key decision criteria in mind. The most important criteria are typically price, item availability, and shipping terms (such as number of days acceptable for shipment and cost), although there are others (e.g., taxes, condition, warranty, etc.). The human user decides on the most appropriate merchant to purchase the item by weighing the various criteria in a manner consistent with the user's needs at that moment. Once a decision is made, the human user uses the online interface of the e-commerce portal to execute the pricing of the order, checkout, delivery and evaluation of the merchant. The context of this use case closely

matches the CBB model and is consistent with the features offered by e-commerce portals.

In the scenarios that are enhanced through the assistance of a transaction agent, the main actors are the human user (i.e., buyer) and transaction agent. The activity of a typical e-commerce portal involves many individual use cases. The illustration in Figure 2 relates them together into an integrated workflow.

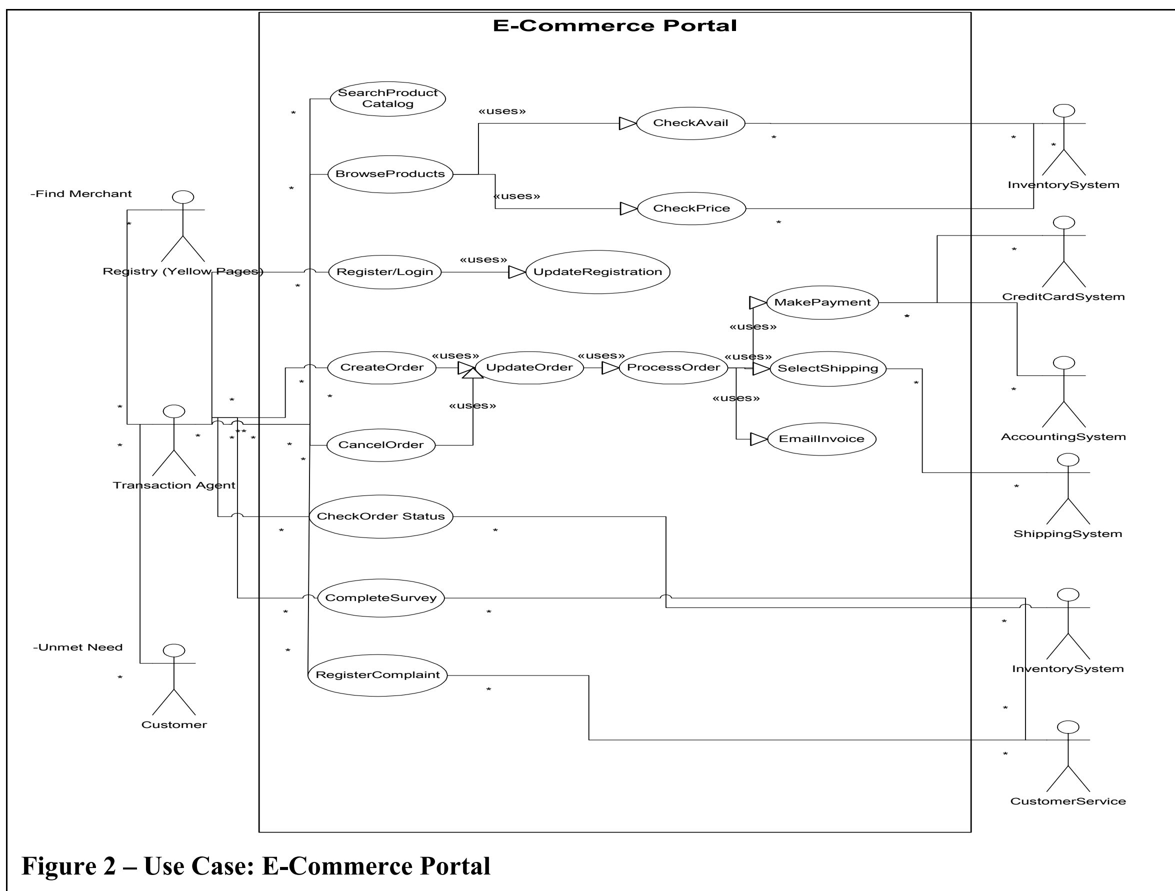

Figure 2 – Use Case: E-Commerce Portal

In these scenarios, the agent acts as a facilitator actor in that it uses the delegated authority from the human user to act on the human user's behalf. The transaction agent's objective is to satisfy the wishes of the human user by determining and executing the set of sub-transactions (i.e., steps) representing the "best" option (i.e., the purchase option that matches the input parameters the best and thus provides the best value proposition).

These agent enhanced use cases can facilitate pull and more complex push (delayed transactions) scenarios, which are described in more detail with examples in chapters 4 and 5.

3.3.2 Software Architecture Overview

As stated, today e-commerce portals are engineered with human users in mind, which is an inhibitor for transaction agents to interoperate with the portal because of the lack of an agent-oriented interface. The transaction agent must navigate to the e-commerce portal and utilize the same HTML interface as the human user, which essentially forces the agent to be coded specifically for a particular e-commerce portal by "scrapping" the page elements from the HTML

To achieve a robust business solution, contemporary e-commerce portals are designed to use both horizontal platform tiers and vertical software layers. This 2-dimensional architectural pattern provides strong internal cohesion that better enables the e-commerce portal to maintain a wide range of system characteristics such as availability, reliability, scalability, extensibility, performance and flexibility.

A tiered architectural style typically consists of four tiers: *client, presentation, business and data*. Tiered software architecture addresses the complexity of the problem from a divide-and-conquer perspective. This strategy divides the overall problem solution into more manageable components and addresses each component with a targeted and specialized architectural solution. For example:

- The client tier supports the needs of the human user and software agent and is the interface for the e-commerce portal. The human typically uses a browser

to access the e-commerce portal while the software agent is a standalone application.

- The presentation tier (or Web tier) processes all HTTP requests. These requests can be GET/POST using static HTML files, JSP or SOAP messages. The Web container administers the JSP life cycle, dispatches service requests to application components, and provides standard interfaces to support session state and information about the current request. Functionally, it generates and presents dynamic content, collects data, controls page flow and maintains state. In essence, it is responsible for the presentation of the application to the end user.
- The business tier contains the business logic, which is encapsulated in the application server consisting of an EJB container with multiple EJBs providing a scalable business logic implementation and database rows' in-memory caching.
- The data tier contains all of the data and supports the environmental needs of a relational database for data persistency.

The second dimension of the architecture consists of the software layers. Typically these layers can be described as: *application, container services, application server services and operating system environment*. Layers are abstractions of the underlying application. They simplify the software architecture by hiding the details of the layers below (e.g., the developer doesn't need to know about the details of load balancing, resource pooling, security, etc.). Like the architectural tiers, the layers provide services to the adjacent layers such that the application (i.e., Enterprise JavaBean

components) is provided services from the container, which is serviced by the application server, which is in turn supported by the operating system environment.

3.3.3 High-Level System Landscape

Figure 3 illustrates the transaction flow between the tiers of an e-commerce portal. It is a typical synchronous request processing model in which the human user requests a service via a browser (usually by clicking on a Web page link) with the e-commerce portal responding in a timely manner. These requests often require dynamic responses and occur in series as a logical transaction where the session must be maintained between them. Java servlets are often used in the front-end of a portal because they provide an efficient and scalable infrastructure for processing HTTP requests, are tightly integrated with the Web server's request processing and enable Java-based session management.

The transaction is triggered by the client submitting an HTTP request for the servlet (or Java Server Page) on the Web server. The Web server forwards servlet requests to a Web (servlet) container (running on the Web server). The Web container is a continually running threaded process that manages the servlet life cycle (creation, initialization, execution, destruction). The Web container runs a Java Virtual Machine to execute the request-processing code, which un-marshals the HTTP request into the appropriate Java objects. The Web container routes the requests to the specified application component (e.g., Enterprise JavaBean). The application component processes the business logic (which may involve routing database queries to the database server) and passes a response object back to the Web container. The Web container then

synthesizes a response in HTML and sends it back to the client, where the human user can view the results in the Web browser.

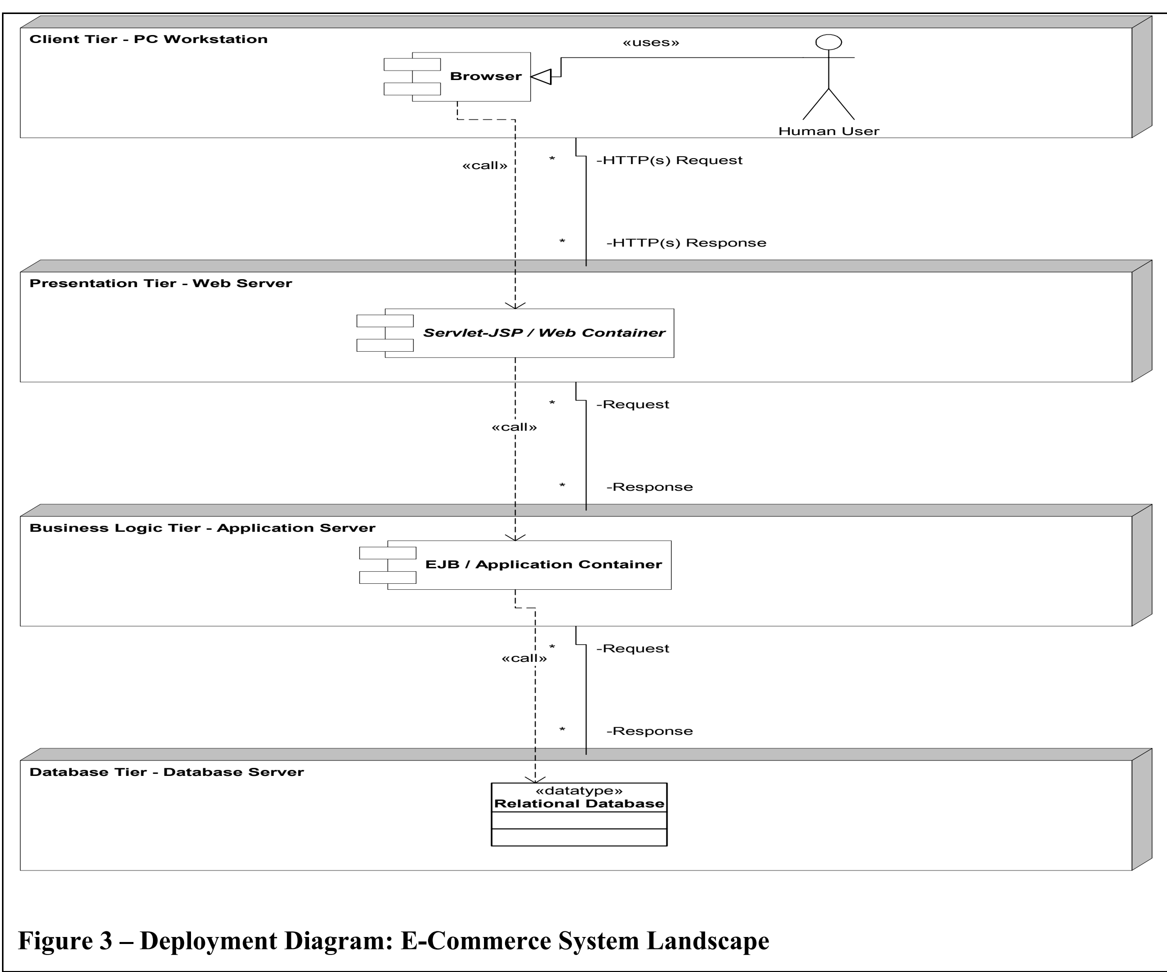

Figure 3 – Deployment Diagram: E-Commerce System Landscape

3.4 Exposing Open Business APIs with Web Services

An important premise of this dissertation is the ability of the transaction agent to communicate to the e-commerce portal through the portal's business API, which is publicly exposed as Web services. This approach provides a route in which change can be minimized to the portal front-end through a one-time support system installation and eliminated for the mission-critical business and database logic back-end tiers. 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, which is solved with a middleware architectural pattern.

Given the network and 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 (i.e. meaning able to interface to each other by design). For this dissertation, the middleware is represented by the client and server-side generic *Agent Interface Support Systems* (see Figure 4). The messaging protocol between the two components must support a wide range of capabilities to facilitate the interaction between the agent and portal. Web services provide many benefits in this regard, which will be highlighted in section 3.4.1.

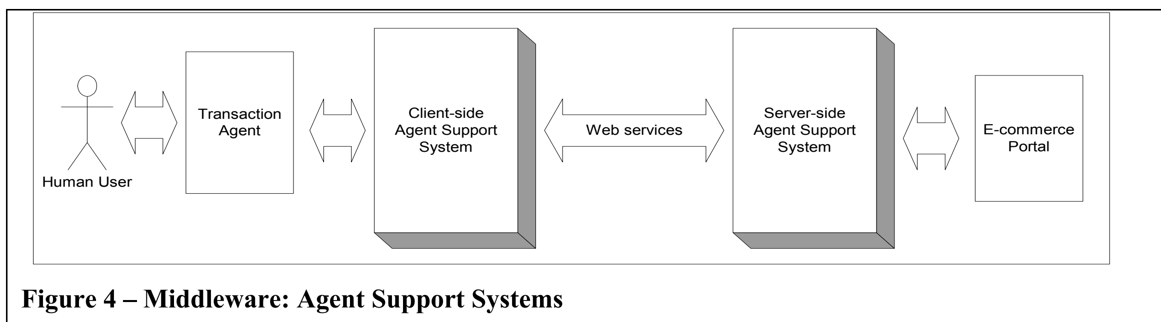

Figure 4 – Middleware: Agent Support Systems

This middleware architecture must natively support synchronous communications, as well as an asynchronous callback process. Transaction agents can operate within this environment in either a pull, push or both modes. When in a pull mode, the request is initiated from the client-side (where the human user and transaction agent reside) and the response comes from the server-side (meaning portal side). In push mode (to support delayed transactions that are described in later sections), the one-way

response is initiated from the server-side (portal) resulting in information (i.e., notification) passed back to the client-side.

3.4.1 Why Web services

Web services are a new approach for integrating systems on distributed heterogeneous platforms. Web services are not accessed via object-model specific protocols, such as the distributed computing solutions (i.e., RMI, CORBA/IIOP, DCOM), but rather through the widely accepted Web protocols. As compared to other distributed computing techniques, Web services provide a layer of abstraction above CORBA or .NET servers and because of this it provides some unique advantages over traditional middleware for the problem highlighted in this dissertation. The design challenges for the solutions described in this dissertation are driven by a set of technical requirements that are supported by Web services. Those technical requirements are:

- Web servers should provide public APIs for agent clients or their supporting systems to call and get information;
- Such calls should not make servers vulnerable to security attacks;
- A comprehensive mechanism should be there to allow the public Web APIs categorized and registered with prominent public servers so client agents or their supporting systems can search for interesting portal APIs;
- The server APIs must support invocations from agents or their supporting systems running on any platform and implemented in any language or technology.

Web services enable enhanced interoperability, better usability and reuse of the back-end portal architecture, and easier deployment using standard Internet protocols and

standards. Functions configured as Web services are platform and language independent, thus enabling a broader range of e-commerce portals accessible by the transaction agent. Since they utilize Web protocols (e.g., HTTP) they benefit from easier firewall compliance (because HTTP typically utilizes port 80), automatic HTTP authentication, encrypted communication via SSL and persistent connections. Also, through the use of several APIs (synchronous and asynchronous), Web services more easily allow the business logic of the portal to be exposed over the World Wide Web, thus leveraging the large investment already made in the back-end of most e-commerce portals.

For transaction agents to achieve the goal of automating e-commerce processes, they must be able to interact with e-commerce portals in a synchronous manner. The common technique used to achieve synchronous interoperability is via a remote procedure call (RPC). Two popular RPC technologies are DCOM (a Microsoft-only standard now incorporated into Microsoft .NET) and CORBA/IIOP. DCOM doesn't support cross-platform interoperability, a major inhibitor for a transaction agent that needs to act in a ubiquitous manner. CORBA/IIOP does provide cross-platform interoperability, but is very complex. Also, solution deployment is more difficult with CORBA because the client must have a CORBA ORB installed, which has financial implications and requires careful version control. Web services solve this problem because the clients only need an XML parser, which has been standardized and publicly available on all major platforms.

Because of these limitations and other beneficial reasons, Web services provide a better RPC architecture. Specifically:

- It minimizes changes required to the back-end e-commerce portal and software agent;
- It provides a loose but synchronous request processing mechanism (i.e., JAX-RPC) for online interactions;
- The messages encapsulated within a SOAP envelop can pass through portal firewalls as normal HTTP traffic;
- Since the requests are sent as XML messages, it supports software agents running on any platform or implemented in any language;
- It enables programmatic access to the e-commerce portals business logic and database tier;
- The JAX-RPC API hides (wraps) the details of the underlying SOAP communications and WSDL descriptions;
- They can be registered with UDDI to aid in the service discovery phase.

3.4.2 Identifying Business Methods for Exporting

In the context of this dissertation, the methods that are in the WSDL define the Web services the e-commerce portal is exposing for transaction agents to interact with. Since the design solution exploits the existing business logic of an e-commerce portal by exposing that logic via an application programming interface, the business logic methods (implemented through EJB or Java beans), which must be defined in the WSDL are the same as those called from the presentation tier to fulfill the transaction life cycle. These methods represent the same methods used by the human user interface.

3.4.3 The Agent-Enabling Interface – Conceptual Design

This section introduces a conceptual design for an *Agent-Enabling Interface* (AEI) using Web services. Within this design, Web services provide loose external coupling that enhances interoperability and access by the transaction agent to the e-commerce portal's business logic through a robust application programming interface (API). The AEI design is based on a service-oriented architecture constructed on a tiered and layered J2EE solution. It affects only the client and presentation tiers, thus maintaining the business and data back-end tiers without change (a stated principle for the dissertation). The Agent-Enabling Interface is designed to utilize complementary Agent Interface Support Systems on the client and server-sides (see Figure 5). The support systems have several components (described in later sections), which achieve the required interoperability between the transaction agent and e-commerce portal.

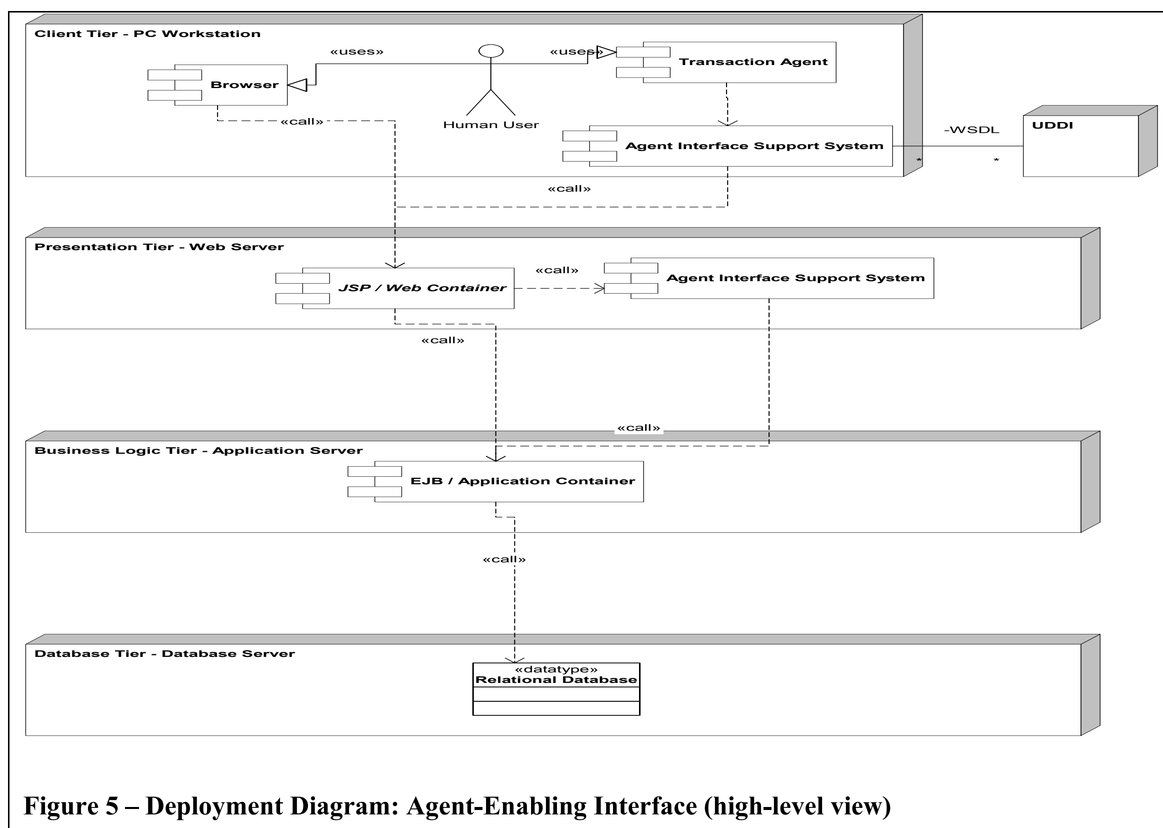

Figure 5 – Deployment Diagram: Agent-Enabling Interface (high-level view)

The AEI design exploits the JAX-RPC API to implement the synchronous communications required for pull mode. The server-side Agent Interface Support System operates on the presentation tier and contains a JAX-RPC service run-time environment and a service endpoint. The client-side support system contains a JAX-RPC client run-time environment, the transaction agent (i.e., the application) and other service components for synthesizing and managing the method calls. The RPC uses SOAP as the messaging protocol and HTTP as the transport protocol. The JAX-RPC client run-time system creates a SOAP message from the remote method call and dispatches the message via HTTP to the service end point (on the e-commerce portal). These Web service components are wrapped with additional functionality (described in later sections and chapters) to achieve the required functionality of the support system.

3.4.4 Generating WSDL files

Web Services Description Language (WSDL) is an XML grammar for describing exposed network services and the specific rules service requesters must utilize when communicating to these Web services via SOAP messages. WSDL service definitions provide the documentation for distributed systems and serve as a guide for automating the details involved in applications communication. It documents the service name, operations that can be called on it, the parameters for those operations, and the location of where to send the requests. Specifically, a WSDL document exposes the method signature, protocol to be used, network address, and data format. Using WSDL is in essence abstractly defining the service functionality and then binding it to a physical protocol. In order for an e-commerce portal to expose its business API, it must describe these methods using a WSDL.

The WSDL structure and syntax is an extensible XML grammar consisting of several different elements (e.g., data type definitions, operations and service bindings) and six parts. It describes network services (i.e., e-commerce portal services) as a series of endpoints that accept messages containing document or procedure oriented information (procedure oriented in this dissertation). The operations and messages are described abstractly and mapped onto one or more concrete transports/encodings. The WSDL will describe a request-response message pattern since the transaction agent and portal interact in near real-time.

To support the AEI, the portal must develop a WSDL that defines each of the interfaces that will be exposed as Web services. Developers generally do not need to understand the structure of the WSDL documents since they are usually generated automatically. Various Web service tools can generate and validate the WSDL documents including xrpcc (available in the Sun Java Web Service Developer Pack), WASP Developer Advanced (Forte for Java) and JBuilder Enterprise to name a few.

3.4.5 Generating Web Services

The WSDL document retrieved from the query of the public registry provides the details of the Web services exposed by the e-commerce portal. It is needed to create the low-level classes that are required for communications between a client and a Web service. A remote-interface will first be defined to enumerate the business methods that are to be exported on the portal. To establish these support system environments, a custom deployment tool would execute on the client and server. This configuration is done as a one-time setup to enable the client to communicate to the portal, as well as run-time when the portal specific E-Commerce Proxies are compiled. The custom

deployment tool is a wrapper around publicly available Web service tools to facilitate the additional tasks required to configure the client and server-side support systems.

On the server (portal) side, the deployment tool generates the WSDL files, ties, servlets, service modules, tables and registers the portal with a UDDI registry. The appropriate Java classes are packaged as a Web application archive (WAR) and deployed on the presentation tier (Web server container). The tie class serves as a proxy for the server-side.

The client-side tool creates the base service modules, tables and Web service components (e.g., SOAP processor). Also, the E-Commerce Proxies (or stub classes) are compiled for each of the interfaces listed in the WSDL returned from the discovery process. A "meta" interface is required by the E-Commerce Proxy because the invocation capabilities of a proxy do not allow it to execute a remote method by executing a text string representing the remote method name and parameter values upon the local stub. Because of this restriction, the invocation must be done through concrete methods created by the deployment tool utilizing the Java class and reflection API framework (described further in chapter 5). The number of parameters and return value for each method is determined from the WSDL. This information is used by the deployment tool to construct the code of this "meta" interface.

The proxy object handles interaction and details of the communications with the remote Web service. It enables the client-side support system to access the Web service methods as if they were methods of a local component. 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 message and send it to the portal'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.

3.4.6 Registering Web Services in Public Registries

Registering Web services in public registries is critical to enable transaction agents to interoperate with e-commerce portals. To advertise its presence, the e-commerce portal would register its service definition WSDL files with the appropriate registry, which provides search facilities and a remote API for accessing the registry. The server-side (i.e., portal) UDDI registration process, which is part of the initial support-system deployment manages the creation, maintenance and deletion of this registration data. The registry transaction workflow is as follows [35]:

- The portal authenticates with the UDDI registry by sending valid credentials (ID/password) to the registry;
- A new organization object is created and populated with data. It contains a Name, Description, ID Key for the organization (UUID), primary contact information, classification, service and service binding objects. Classification codes are managed as taxonomies by 3rd party organizations (e.g., NAICS) to categorize a business into various industries;

Services and service bindings are added to the organization. A service object has a name, description and unique key, which is generated by the registry when it was

registered. It may also have classifications associated with it. A service typically has service bindings that describe how to access the service. A ServiceBinding object usually has a description, URI and a specification link that provides the linkage between the binding and the technical specification.

3.4.7 The Agent-Enabling Interface – Components Description

As defined, the AEI requires complementary support systems on the client and server. The support systems consist of service components responsible for the functionality encapsulating the Web services. These services are used by the transaction agent operating in pull mode, but also support the agent in push mode.

The client-side support system consists of the *Generic Middleware between Agents and Portals (GMAP),* which utilizes a component and table-based design to implement interoperability between the transaction agent, public UDDI registry and e-commerce portal. GMAP provides the infrastructure and integration supporting the major generic service functions: the *Registry Query Manager, Portal Invocation Manager* and *Callback Web Service.*

For support of pull mode, the server-side support system is accessed via a simple Web service represented by two interfaces for invoking the portal-specific business logic and retrieving the *ActionWords object* (created by the portal operator which contains the portal method name and signature mapping to an action word) Since the invocations are using HTTP on its standard port (i.e., 80), these interfaces act as entry-point servlets to the tie objects, which invokes the method on the intended remote service.

In the solution design (see Figure 6), an Agent Interface Support System operates on the client tier and is responsible for registry query services and SOAP processing for

RPC processing. Identification of e-commerce portals is achieved through a Registry Query Manager, which via a discovery process finds e-commerce portals through a query of a UDDI repository. 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 table* 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.

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 *ActionWords object* for closing the semantic gap between the transaction agent and the portal. The Portal Invocation Manager will maintain the proxy object and the ActionWords object for the duration of the transaction and populate a *Method Description table* for the Web service methods of that portal (using the WSDL

and ActionWords object). In this table, each method is mapped to a standardized action word 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.

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 returned 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 ActionWords object to identify the accessor methods of the returned object, and use them to retrieve the returned value components, populate them into a *Results table*, and qualify each of these value components with description keywords contained in the ActionWords object (see chapter 5 for details). When all the return values for the current sub-transactions have been populated into the Results table, the table will be accessed by the transaction agent for processing.

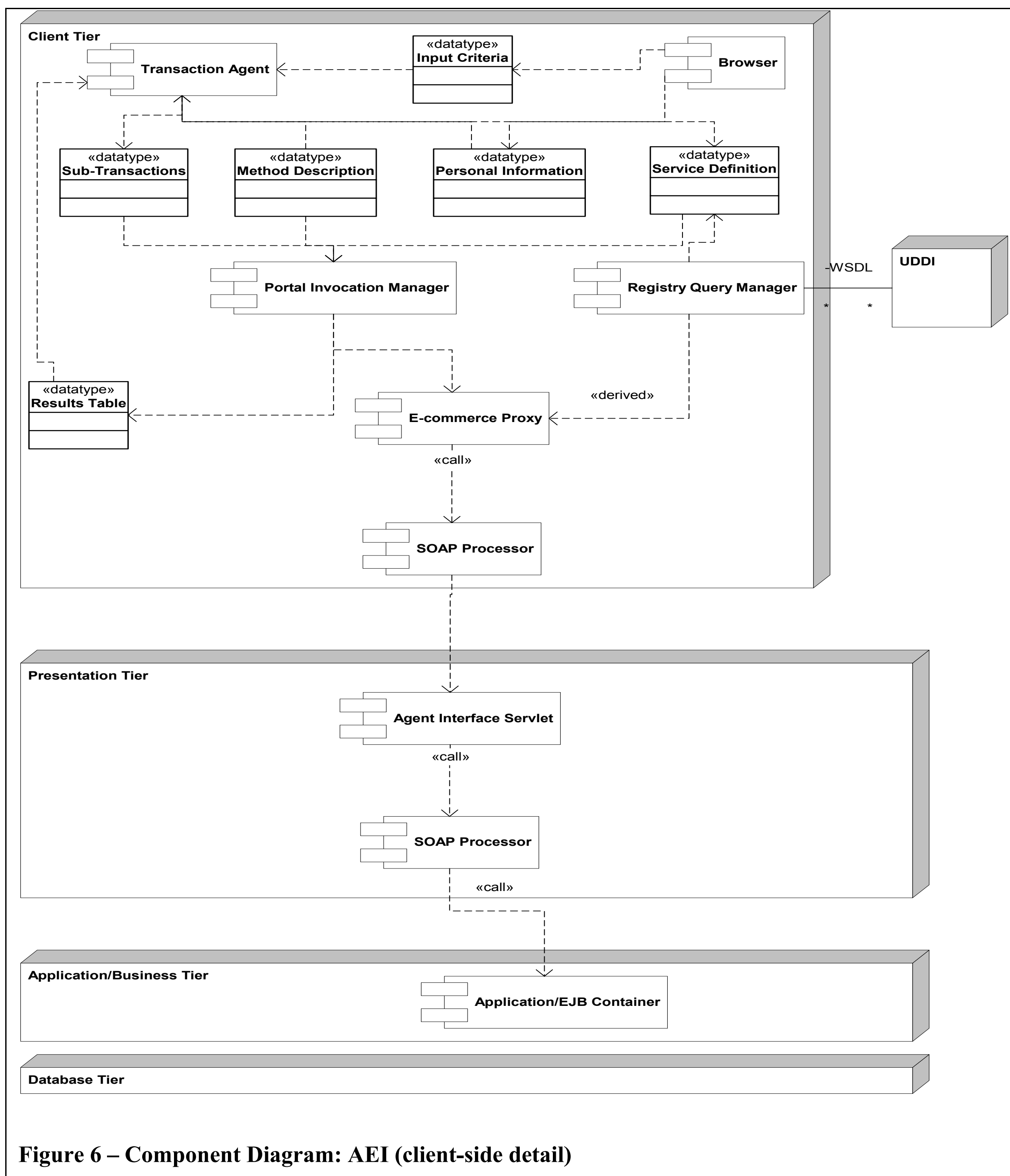

Figure 6 – Component Diagram: AEI (client-side detail)

3.5 Supporting Delayed Transactions with Server Callbacks

The previous section described an interface design that met many of the objectives stated for this dissertation. Web services were utilized and substantiated as important elements of a solution for software agent interoperability. This section evolves that design into a generalized software architecture by adding an additional software tier

to support an efficient process for delayed transactions. The ability for a transaction agent to react to changes in its environment (i.e., reactivity) is an important characteristic of the agent in achieving its mission. To that end, in order for the transaction agent to efficiently react to changes in its environment (e.g., a change in the state of the data), the e-commerce portal needs the ability to "callback" to the software agent to communicate relevant information updates. Since Web services doesn't support a callback feature (unlike CORBA), this functionality must be designed into the software architecture of the solution.

3.5.1 Value of Delayed Transactions

Delayed transactions can be important in the context of e-commerce. The duration of a transaction for e-commerce is typically much longer than the corresponding traditional retail transaction. Given the nature of the shopping experience, an e-commerce transaction may take a longer duration as the human user takes advantage of virtual shopping (i.e., trying to meet a product need without visiting retail stores). Retail transactions because of their physical nature are more likely to occur in a snapshot in time (i.e., typically the human consumer visits the retail outlet, purchases and takes possession of the item). Since e-commerce transactions may occur over a time span, key decision parameters may change affecting the value proposition of the transaction. In order to provide an opportunity for the human user to take advantage of these changes affecting the value proposition, these critical changes need to be communicated back to the transaction agent for its consideration.

There are two approaches to initiating event communications between service providers and requestors (or suppliers and consumers): pull and push. In the pull

scenario, the transaction agent would be required to poll the various e-commerce portals it is attempting to communicate with (therefore there must be a tracking feature as part of the client-side supporting system for tracking all current portals the transaction agent is interacting with). This polling operation would need to be executed on a frequent basis to ensure an opportunity isn't missed to transact with parameters that improve the value proposition. This is inherently inefficient because in most scenarios key decision parameter data isn't likely to change, but the cost of a missed opportunity may be high. Therefore, it is a high cost for a low probability of return. The contrary approach is a push operation. In this case, the e-commerce portal would have each transaction agent registered with the particular start/end dates/times in which the transaction may occur in. During this period, if any change occurs with the key decision parameters, the e-commerce portal can message the transaction agent via a callback service. This is a more efficient model since callbacks are only triggered on actual changes to key decision parameters. The model is essentially a polling operation between the Callback tier and Database tier and could be made more efficient if the callbacks were event driven triggered by changes of state in the Database tier, but that would require changes to the back-end of the portal which conflicts with a main principle of this research. This model requires components for registration, scheduling and checking in the server-side support system.

3.5.2 Server Callback: Unsupported by Web Service Technologies

The current specifications for Web services do not support a server callback. Web services, unlike other RPC-oriented middleware, use unidirectional asynchronous messaging. At its core, SOAP is fundamentally a one-way communication specification

between the sender and receiver, which contains methods for adapting its one-way messaging for the request-response convention typical of RPC-oriented communications. Thus, Web services are fundamentally one-way, asynchronous messages mapped onto executable software programs and do not contain the architecture elements for logic capable of enabling server callback.

What is needed is more than simple server callback. Because the portal usually serves many concurrent transaction agents, and the events that the agents are waiting for may not happen for extended periods of time, the server callback mechanism must be decoupled from the portal business logic. What is needed is an extension to the basic Web service architecture to provide this functionality. This extension can be fulfilled with an additional architectural tier that is portal-independent and implements many of the functions of the CORBA event service (e.g., push client registration and maintenance, callback criteria parsing and checking, and client callback). While a CORBA event object receives event updates from external event sources, the designed solution needs to schedule event checks and proactively check potential portal events for the remote push clients. This proactive portal event checking is critical in minimizing the changes to the portal business logic tier and improving portal performance by checking only those portal events that are interesting to the currently registered push agents.

3.5.3 Enhancing the Contemporary Web Architecture with Callbacks

The main functions required for supporting transaction agents in push mode (i.e., delayed transactions) require a more complex server-side support system. In this case, the agent can register itself as a push client with the portal using an exposed registration function, and receive callbacks from the portal when the portal has information satisfying

the agent's criteria. Parameters to identify the attributes to be monitored, the frequency of data checking and terms defining conditions for state changes, as well as the identifier for the sub-transaction are passed as part of the registration and stored in the *Callback tier* (see chapter 4 for details). In addition, a WSDL is passed as part of the registration process to define the Web service the client-side is exposing to the e-commerce portal for the Callback Web service.

As described, the Agent Interface Support System will facilitate the agent registering itself as a push client on a portal, and receive callbacks from portals when the portal has information satisfying the agent's criteria (see the callback related service functions and tables in Figure 7). This is achieved by services running on the client and server-support systems, as well as exposed APIs. The client-based E-Commerce Proxy makes a remote method call of the Callback Registrar service that registers the transaction agent with the portal, which is stored in a server-based table along with the various attributes, checking frequency and condition parameters passed as part of the call. These attribute parameters define what data is to be monitored for state changes, the frequency of checking for state changes and the conditions for callback, which enables a level of precision to be set around the amount of state change (e.g., price changes over a certain amount). In particular, the transaction start and end dates are critical since they are used by the Callback tier as a filter to know what duration of time the callback period is valid.

The Callback tier supports a scheduling and checking function that utilizes the frequency parameters passed during registration to poll the database tier for changes in the parameters satisfying the conditions set. If a callback is required, it will generate a

callback notification to the corresponding Callback Web Service through its proxy object. The GMAP will use the returned sub-transaction identifier to activate the proper method invocation through its Portal Invocation Manager to retrieve the latest portal data.

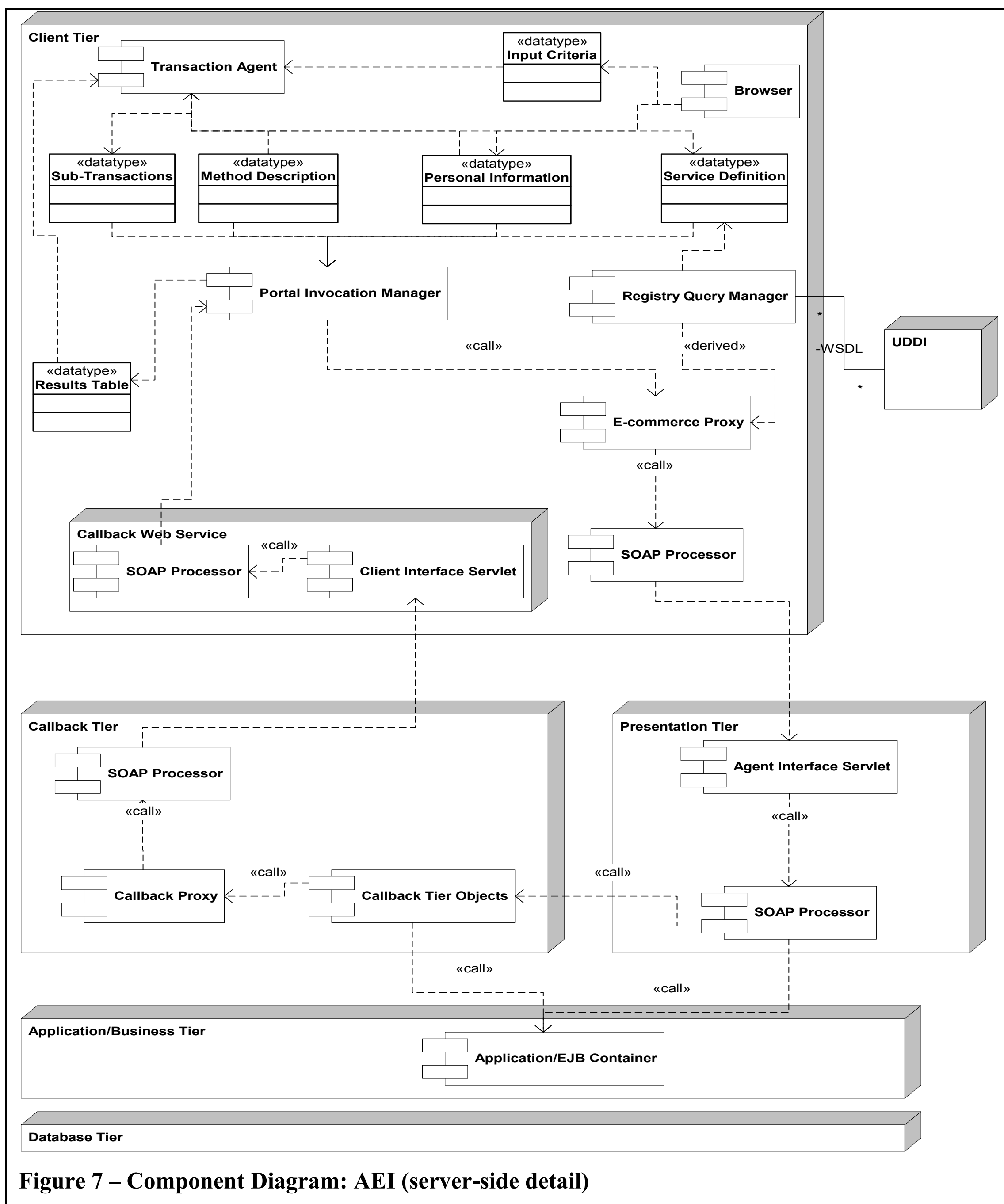

Figure 7 – Component Diagram: AEI (server-side detail)

3.6 Summary

The chapter describes several important design constructs that are defined at a conceptual-level, which in combination, act to enhance transaction agent interoperability with e-commerce portals. The proposed approach can be applied to any existing e-commerce portal, without any changes to its existing functions, to provide an open API for client-side transaction agents to access the same business logic that human clients can through a Web browser. It uses middleware technology to enable network-blind transaction agents to interact with the portals in the form of client and server-side Agent Interface Support Systems. The support systems communicate via the Agent-Enabling Interface, which utilizes Web services to support a broad array of e-commerce portals.

This chapter also contributes the design of two generic and reusable software components: the GMAP for supporting transaction agents on the client side, and the Callback tier for supporting the callback notifications from a portal to a transaction agent. The proposed action words based semantic translation system provides the transaction agent researchers a clean interface to the agents' complex and technology-rich environment.

Chapter 4 – Enabling Portals to Work with Pull/Push Agents

This chapter provides detailed architectural design and implementation considerations for the support of callback for push agents. The solution exploits the advantages of Web services and evolves into a generalized software architecture that supports push agents by adding an additional software tier (i.e., Callback tier) to support an efficient process for delayed transactions. The specific API will be described including the generic functions, method signatures and available options to enable the agent/portal interoperability. In addition, a logical design for the components required for the Callback tier is described along with integration strategies for implementing them. Lastly, use cases and work flow activity diagrams are used to illustrate examples of the proposed design solution satisfying the needs of typical callback scenarios.

4.1 Generic Functions Supporting Pull/Push Agents

We call a transaction agent a *pull agent* if it needs to pull information from a portal through synchronous remote method invocations, and the portal has no means to inform the agent of any data change voluntarily.

We call a transaction agent a *push agent* if, in addition to pulling portal information through synchronous remote method invocations, it can also receive notification from the portal of any data changes.

In theory a pull agent should be enough. To check the data change on a portal, a pull agent can repeatedly pull portal data. But this will lead to unnecessary heavy network traffic, unnecessary heavy CPU workload on client-side, as well as an unnecessary heavy workload on the portal. For example, let us assume many clients are

interested in tracing the price of a very dynamic and important stock. The portal will be flooded with repeated inquires about the current value of this stock, and the scalability of the portal business logic will be seriously challenged. By adopting a push agent approach, each client interested in that stock will only register with the portal once, and the portal can group multiple clients' inquires and schedule stock value database queries, which can be shared by all of the interested clients. Therefore, push agents could greatly improve portal efficiency, client efficiency, and network efficiency.

While Web service technology can support pull clients directly through client invocation on portal Web services, it doesn't support server callbacks. This is an architectural limitation of Web services. In this chapter, we will introduce a Callback tier to provide the callback function in the Web service environment, and support efficient push agents.

4.2 Assumptions for the Generic Callback Tier Design

The design of the Callback tier is based on several assumptions. Those assumptions are:

- Each client registration will only refer to one portal datum name (i.e., attribute to monitor), although the agent can register for multiple callbacks. Each client registration can also optionally specify the frequency for the Callback tier to check the value of the interested portal data. It is assumed that the client check frequency requirement is satisfied as long as the actual data checking frequency is greater than or equal to the one the client specified (checking data more frequently than asked by a client should not cause a

problem for the client transactions). Establishing a cost for the checking frequency would be a good practice in order to client balance demand for up to-date checks with the supply of server computing resources, but that is not explored in the dissertation;

- The client environment will maintain a unique sub-transaction identifier (ID) for each of its portal callback registrations, and maintain all information about these callbacks in rows of a client-based table. During client callback registration, this unique ID will be passed to the server-side Callback tier. During callback, this ID will be passed back to the client to identify the data attribute being monitored and trigger a pull method call to retrieve the updated data;
- The callback service is triggered by changes in state of monitored attributes that exceed conditions defined in the registration process; no other prioritization scheme to deal with scarce products/items or preemptive callbacks is provided for in the design (pull mode allows for immediate response);
- Each client will implement a Callback Web service implementing a standard callback interface, detailed in section 5.2.3;
- The portal should be designed in a tiered and layered architectural style so that separation of function is clearly defined. This is critical since the Callback tier is positioned at the front of the portal architecture (along with the presentation tier) and must be able to effectively interoperate with the back-end tiers;

- On the portal, all database inquiries are through business logic invocations. If the portal supports application servers, then these business logic methods typically corresponding to EJB entity bean methods. Otherwise they can be methods of a wrapper class centralizing and synchronizing database accesses (since this is always considered a good practice, this is not a limiting assumption). For each method for checking portal data (like checking prices for items), it has a unique action word corresponding to a method name and a unique method signature like "double checkPrice(bookISBN)." These action words are mapped to the method names in an object in the Callback tier.

4.3 Callback Tier: A New Generic Web Application Tier for Push Services

The Callback tier represents an evolution of the traditional n-tier design (client, presentation, business and database) commonly used for e-commerce portals. It executes several services in support of the client-side transaction agent, specifically registration, scheduling, database query/checking and agent callback. The registration API is called from the client-side support system and is thus exposed as a Web service.

For portals required to support special callback criteria, the Callback tier can be custom designed. But for standard portals, here we provide a generic Callback tier design. This generic callback is intended to be implemented as a reusable software component, and it can be easily adapted to work on existing portals to support push agents. The design for this generic Callback tier can also be used as the validation of the approach, and provide guidance to the design and implementation of a customized Callback tier. A high-level description of the Callback tier functions follows.

4.3.1 Registration

A transaction agent can, through its supporting environment, register as a push agent with a portal. The portal will support a new registration Web service, which will be defined in a portal WSDL file that is published through UDDI (the entry point *Agent Interface Servlet* is generated during the deployment process and resides on the presentation tier). A push agent will make a Web service invocation to register itself. During registration the agent will pass several parameters including a client assigned subtransactionID that identifies the sub-transaction on the client, a client WSDL file of its Callback Web service, which encapsulates descriptions of client IP address as well as available callback methods, and other registration specific parameter values.

The Callback tier will first validate registration information, which includes checking whether the provided client WSDL file can be used to generate a *Callback Proxy*, and whether the callback criteria are referring to valid server data and ranges. The registration information will be stored in a database, which will provide persistency to the registration data to avoid data loss due to server failures. An in-memory *Callback table* will be used to store light-weight information to perform callbacks where each row contains a primary key to the corresponding database entry, a reference to the Callback Proxy (generated during registration validation), and a reference to the *CallbackCriteria* object. Each CallbackCriteria object contains the name of one portal's datum name (like stock name/ticker symbol, book name/ISBN, etc.), the allowed condition value, checking frequency, etc. This in-memory table, as well its related database table, will be updated upon agent registration, as well as updated or cancelled registrations.

4.3.2 Scheduling

The Callback tier will have a *Callback Scheduler* to schedule business logic invocations for all the registered callback agents. Multiple such agents may need to check the same portal data, and they can be replaced by a single call. The input to this Callback Scheduler is the information in the in-memory Callback table described above, as well as the CallbackCriteria objects. The output of the Callback Scheduler is an in-memory *Schedule table*. It is created in a two-pass operation. During the first pass, a table is created that contains one row for each business method invocation to check the value of one portal datum name. Each row contains (1) its corresponding callback table index, (2) method name, (3) method argument, and required data checking frequency. This table will then be sorted primarily according to method name and argument, secondarily according to frequencies. A new Schedule table will be generated, one row for each unique pair of method name and argument. For each such row, a checking frequency will be specified that is the maximum frequency found during the previous sorting; and a list of indexes for the in-memory Callback table will also be specified, which are the callback indexes for all the rows (after sorting) that share the same values for method name and argument. The resulting Schedule table will be passed to the Callback Checking objects.

4.3.3 Checking

The Callback tier will have *Callback Checkers* to carry out the scheduled portal data checks by calling server-side business methods. Each Callback Checker object will run in a separate thread and be responsible for checking the value of one portal datum item. It will maintain an index to the Schedule table for its scheduled business logic method call. It will sleep for that row's delay time (i.e., the inverse of frequency); then

call the specified business logic method. It will then follow each listed index for the in-memory Callback table and compare the current results to the previous call results and check whether the data change should trigger a callback notification to the client/transaction agent (the previous value can be stored in a wrapper object for this purpose). The callback notification is triggered only in the case where a change satisfies the condition statement stored in the CallbackCriteria object. If callback is needed, then the callback to the agent is made through the stored CallbackProxyReference in the Callback table.

4.3.4 Client Callback

A callback to the client/transaction agent is triggered by a change in state satisfying the condition statement, which is stored by value in the CallbackCriteria object. To execute the call, the Callback Checker formats the method call signature, transforms it into a SOAP JAX-RPC, passes it to the client's entry point Client Interface Servlet and SOAP processor, which parses the SOAP message and executes the `callBack` method that triggers the GMAP to process a synchronous method call to update the client-based Results table. The Client Interface Servlet exposes a Web service and runs in a Web container on the client. The details of the API are described in section 5.2.3.

4.4 Callback Tier: Application Programming Interface (API)

The API for the Callback tier will support the publicly exposed registration function, as well as the internal scheduling and callback functions.

4.4.1 Registration Interface

To make a Callback tier generic and implemented as a software component individually deployable on an e-commerce portal and reachable by all remote agents, the

interface for push agent registration needs to be standardized. The remote agent registration interface is proposed in the form of a Java interface:

```
public interface PushAgentRegister   {
        public int addRegistration(…);
        public int changeRegistration(…);
        public int deleteRegistration(…); }
```

Figure 8 – Registration Interface

addRegistration

`public int` ***`addRegistration`*** `(int subtransactionID, String clientWSDL, CallbackCriteria keyData);`

Description:

This method registers the transaction agent with the e-commerce portal to trigger a portal callback in the event of state changes in any of the key data criteria that satisfy the conditions statement. This information is stored in the server-side persistent Agent Registry database and in-memory Callback table resident on the Callback tier, as well as a CallbackCriteria object.

Parameters:

- `int subtransactionID` – The subtransactionID uniquely identifies the sub-transaction that originated through the registration. It is created and stored on the client-side support system and passed as part of callback registration.
- `String clientWSDL` – The client-side support system passes the Web Service Description Language file, which defines the Web services exposed by the client-side support system for server callback. It encapsulates the client IP

address, callback methods, method signatures and other details to compile a server-side Callback Proxy.

- CallbackCriteria keyData – It is an object that contains encapsulated attributes used to manage the callback process. The specific encapsulated attributes are:

    - `String methodName` – It is the actual name of the portal method to check values of a particular type of portal data (i.e., price). It is mapped to the ActionWord (see section 5.3.5) and stored in the ActionWords object on the portal and passed to clients through a Web service API executed by the client-side support system.

    - `Vector dataName` – The dataName vector are the parameters for the methodName. If it is not needed, it will be represented by null.

    - `String condition` – The conditions attribute identifies the Boolean conditions for triggering a callback. The conditions are expressed as statements that will be used by the Callback Checker to filter results. The statements are expressed in BNF in the following manner:

      <rel-oper> ::= '=' | '<' | '>' | '<=' | '>=' | '<>'

      <bool-oper> ::= AND | OR

      <digit> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0

      <number> ::= <digit> | <digit> <number>

      <float> ::= <number> | <number> '.' <number>

      <signedfloat> ::= '-' <float> | <float>

<term> ::= '(' <rel-oper> ',' <signedfloat> ')'

<term-seq>::= <term> | <term> ',' <term-seq>

<condition> ::= <term> | '{' <bool-oper> ', {' <term-seq> '}}'

For example, a statement representing the condition of 10 < data <= 100 is: {AND, {(">", 10), ("<=", 100)}}, meaning that is has a sequence of two conditions; these two conditions must both be true (i.e., "AND") for the portal callback to be activated.

- `Date startDate` – The start date/time of the callback period.
- `Date endDate` – The end date/time of the callback period.
- `int frequency` – An integer representing the frequency per hour for checking the portal back-end for any state changes.

The partial code definition for the parameters in the class is as follows:

```
public class CallbackCriteria    {
     private int subtransactionID;
     private String methodName;
     private Vector dataName;
     private String condition;
     private Date startDate;
     private Date endDate;
     private int frequency;
                                 }
```

Figure 9 – CallbackCriteria Class

Returns:

A server-side callback identifier number (serverCallbackID)

Error codes:
-1 Agent already registered
-2 Failure in the server-side registration table update
-3 The Callback tier failed to ping the client-side Callback Web service

* * *

changeRegistration

```
public int changeRegistration (int serverCallbackID,
CallbackCriteria keyData);
```

Description:

Modifies the registration of the key criteria data originally passed with the addRegistration call.

Parameters:

- `int serverCallbackID` – The server-side callback ID created and returned to the client by the `addRegistration` method call.
- `CallbackCriteria keyData` – Same definition as when used for `addRegistration`.

Returns:

0 if succeeded or negative error code if failed

Error codes:
-1 Agent not registered
-2 Failure in the server-side registration table update
-3 The Callback tier failed to ping the client-side Callback Web service

* * *

deleteRegistration

```
public int deleteRegistration (int serverCallbackID);
```

Description:

Cancels the registration premature to the date range expiring. This function will delete the relevant rows from the Agent Registry, in-memory Callback and Schedule tables, as well as delete the appropriate CallbackCriteria object.

Parameters:

- `int serverCallbackID` – The server-side callback ID created and returned to the client by the `addRegistration` method call.

Returns:

0 if succeeded or negative error code if failed

Error codes:
-1 Agent not registered

4.4.2 Scheduling Method

createSchedule

```
CallbackInmemorySchedTable createSchedule (Object
CallbackInmemoryCBTable);
```

Description:

When a client adds/modifies/deletes a callback registration, the Scheduling function will be invoked. This function will read the contents of the in-memory Callback table and CallbackCriteria objects (referenced in the Callback table) and create an output Schedule table.

Parameters:

- `CallbackInmemoryCBTable` – As defined in section 4.5.2.

Returns:

An in-memory Callback Schedule table in which there is one row for each pair of methodName and dataName to check the value of one portal datum. Each row will contain the name of the portal dataName, the portal methodName for checking the portal data, the time delay for the next check that the Callback Checker will invoke the method and the indices to the in-memory Callback table rows for those agents interested in this portal data.

4.5 A Logical Design of the Callback Tier

The logical design of the server-side support system is split between the presentation tier and a newly defined Callback tier. The Callback tier is a generic component of the server-side support system illustrated in Figure 10. The major design features of the Callback tier are the services (*Callback Registrar, Callback Scheduler* and *Callback Checker*, as well as a *Callback Proxy*, which utilizes a SOAP processor for processing callback messages back to the client-side support system). These components work in combination to support transaction agents in push mode.

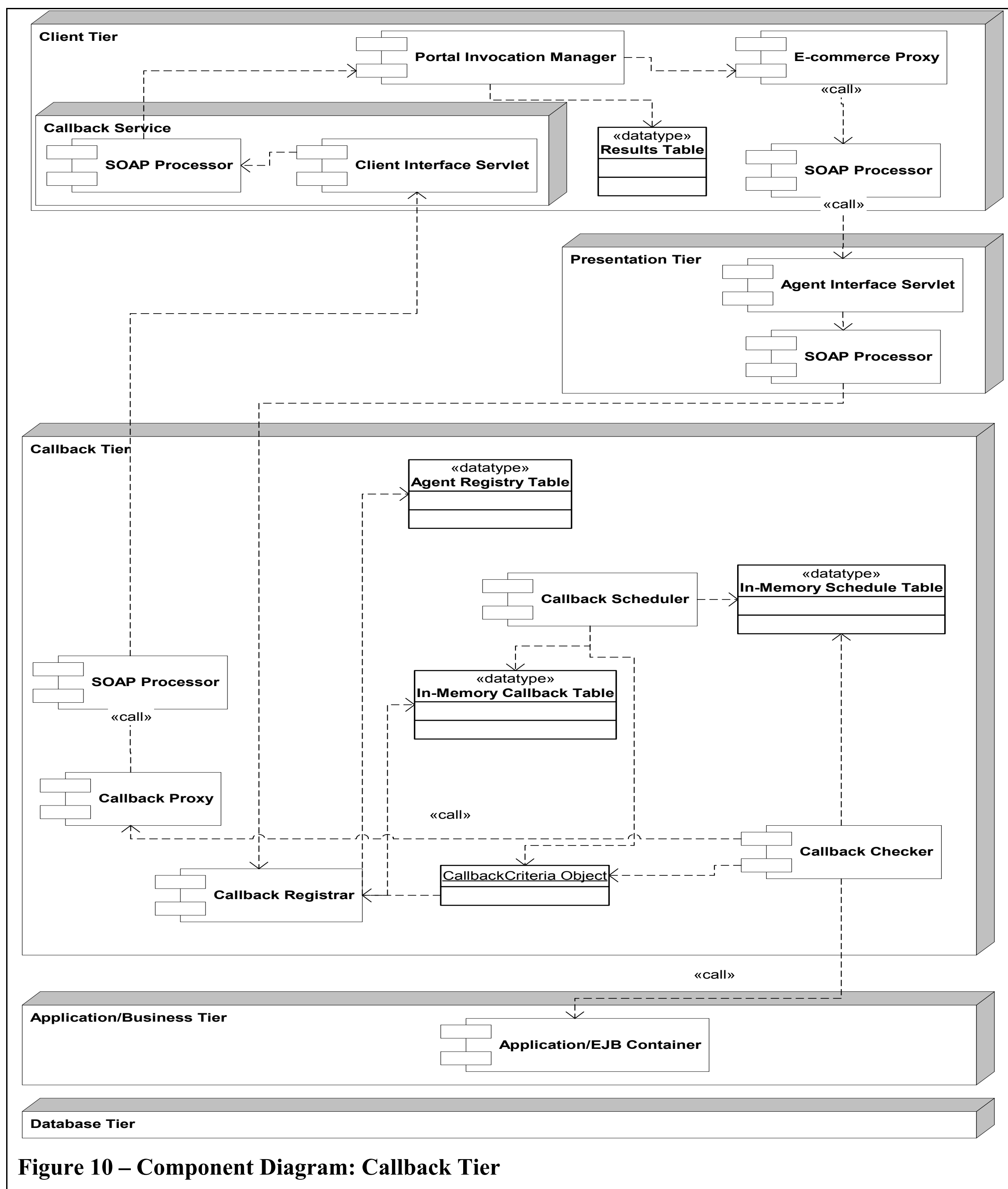

Figure 10 – Component Diagram: Callback Tier

4.5.1 In-Memory Callback Table

The Callback table is represented as an in-memory data structure. It is maintained as part of the registration process and stored persistently in the Agent Registry table. It has one row per each registered agent callback. The agent can register for multiple callbacks. The in-memory table contains the following attributes:

Attribute	Description
PrimaryKey	• A primary key to the persistent Agent Registry database table.
CallbackProxyReference	• A reference to the callback proxy generated during registration.
CallbackCriteriaReference	• A reference to a CallbackCriteria object. Each object contains the name of one portal's methodName and dataName arguments. Other data stored in the object include attributes to monitor, a Boolean condition, checking frequency, etc.

Table 3 – In-memory Callback Table

4.5.2 In-Memory Scheduling Table

The Scheduling table is represented as an in-memory data structure. It is created in a two-pass operation and maintained as part of the scheduling function. The table has one row for each unique pair of MethodName and DataName attributes. The in-memory table contains the following attributes:

Attribute	Description
MethodName	• The name of the method on the e-commerce portal stored in the ActionWords object.
DataName	• The parameter value(s) to pass to the method that reads the back-end database.
TimeDelay	• The time delay (in seconds) between consecutive invocations to a corresponding business logic method.
Indices	• The indices to the in-memory Callback table rows for those callback agents interested in this portal data value.

Table 4 – In-memory Schedule Table

4.6 Integration Strategies of Callback with Legacy Portals

In order for an existing e-commerce portal to utilize the generic Callback tier, it must be integrated and deployed as a tier within the infrastructure of the portal. Those integration and deployment strategies are:

- Use a Web service tool to read the PushAgentRegister interface, generate a WSDL file for its three registration methods, as well as one servlet working as an endpoint on the portal for callback registration, and compile the classes required;
- The business methods must be identified (see section 3.4.2) and their names must be populated into the ActionWords object (details in chapter 5);
- Complete the portal WSDL file with all definitions of methods that are exposed for the back-end business logic;
- The client-side Callback service definition (WSDL) must be generated using a Web service deployment tool;
- The portal must be registered via UDDI with key words established as part of the UDDI data model to assist in the client-side discovery process.

4.7 Typical Use Cases and Workflow: Push Mode

The ability for the solution to support callback is important to handle the use cases associated with delayed transactions in which the transaction agent reacts to changes in its environment. For example, suppose the human user does not need to fulfill a particular need immediately and would prefer to "shop virtually" on the belief that over time prices might drop (which is often true of computers, peripherals and personal electronics). In this use case, the human user would establish criteria that instruct the transaction agent to establish a delayed transaction relationship with each of the appropriate portals located in the registry search.

In these use cases for push agents, more complex scenarios can play out because of the delayed and asynchronous nature of the process. These use cases are preceded by

a registration process in which the transaction agent establishes a callback relationship with the e-commerce portal for a specific time frame. For instance, when purchasing a typical consumer product, price and availability are critical criteria and call backs would be established based on these attributes. When purchasing financial instruments (e.g., stocks) where time may be of the essence, again price is critical, but the frequency parameter for scheduling back-end queries is equally important and would be set as part of the registration process. All of these use case scenarios are handled by the design solution for the Callback tier in which a notification is given to the client-side Agent Interface Support System to retrieve the latest value of the data attribute being monitored.

To illustrate the concepts, a scenario in which three separate agents/clients register for callback to monitor the price of a few stocks with the ultimate objective to purchase the stock (the purchase use case scenario is described in detail in chapter 5). In addition to the typical input parameters, a duration time frame (e.g., start and end date) is provided so that the transaction agent is given a set period to execute the work. Given this time span, information pertaining to the purchase options may change, which could affect the value proposition for the purchase. The main actors are the human user (i.e., buyer) and transaction agent. This example illustrates the workflow associated with the Callback tier for a stock price callback, but given the generic nature of the solution, it can be broadly used for many types of e-commerce. The workflow associated with this use case and use of the Callback tier resources is documented in Table 5 (some details were described in earlier sections).

Workflow
1. The human user for Agent1 (A1) creates a business transaction by entering criteria pertaining to the item(s) of interest. In this scenario, the item identifier (stock symbol), pricing condition (Agent1 has an option to buy if the price is less than 90), start/end date for the callback period and frequency to check for price changes is entered as the business transaction. The agent parses the business transaction creating individual attributes and this triggers the discovery process from the client support system, which creates a variable number of client-based E-Commerce Proxies. From the parsed input criteria, the transaction agent creates individual sub-transactions (stored in a client-side Sub-transactions table) that represent the individual steps required to achieve the business transaction (described in chapter 5). The client-side support system creates a SubtransactionID that identifies this particular transaction for callback. The criteria used for creating the sub-transactions are stored in an object, which is passed to the portal during registration (CallbackCriteria).
2. Likewise, following a similar workflow, a second human user with Agent2 (A2) registers for the same stock symbol, but a different price condition and frequency. A third person with Agent3 (A3) registers for a different stock symbol with a corresponding price condition, as well as a shorter callback period and higher checking frequency. The criteria for each of the agents are noted at the top of the activity diagram (Figure 11)
3. On the server-side support system, the registration process has un-marshaled three CallbackCriteria objects representing the parameters passed from the clients during registration. The process has also executed several validation tasks (e.g., setting up the Callback Proxy). In addition, it updated the in-memory Callback table, creating a row for each agent with a PrimaryKey reference to the persistent database table, as well as references to the Callback Proxy and CallbackCriteria objects.
4. The Scheduling function (createSchedule) is invoked from the registration process and it utilizes the Callback table and CallbackCriteria objects to create an in-memory Schedule table in a two-pass operation (as described earlier). It contains one row for each business method invocation to check the value of one portal datum. The structure of this table enables a design efficiency, which allows only one back-end database check for multiple registered agents when they have criteria in common (i.e., performing the same task as denoted by the MethodName on the same DataName). The Frequency parameter is converted from checks per hour to a duration time (in seconds) between consecutive checks.
5. For the resulting sorted table from the first-pass of the scheduling algorithm, Agent3 is checking 5 times per hour or once every 12 minutes, which is 720 seconds after the start date. Agent2 requires a check every 900 seconds and

	Agent1 every 1800 seconds. To create a more efficient schedule, a second-pass operation occurs, which creates a new Schedule table with one row for each unique pair of MethodName and DataName and a checking frequency that is the maximum frequency. So, in this case, since Agents 1 and 2 share the same MethodName and DataName, the resultant Schedule table has fewer rows, and the performance benefit is that the price check for the IBM stock can be shared by Agents 1 and 2 (eliminating unnecessary back-end database checks).
6.	Each of the Callback Checker objects runs in a separate thread and is instantiated as part of the Scheduling function. It maintains an index back to the Schedule table and sleeps for the time delay (e.g., 720 seconds for Agent3 for the first row). It then calls the business logic method by using the MethodName and the DataName (i.e., BID), which is substituted into the method call, e.g., checkPrice ("BID"). The price results are returned and the Indices from the Schedule table are used to lookup in the Callback table the CallbackCriteria references in order to retrieve the Condition parameter.
7.	If the callback condition is satisfied (i.e., a state change has occurred and the price of BID is less than 10), then a notification callback to the client/transaction agent is executed. Following the previous example, if the second transaction resulted in a return price for IBM of 93, then only Agent2 (less than 95) would trigger a callback to the appropriate Callback Proxy, since the price did not satisfy the condition statement set by Agent1 (less than 90).

Table 5 – Delayed Transactions Workflow

The activity diagram for matching this use case and workflow is illustrated in Figure 11.

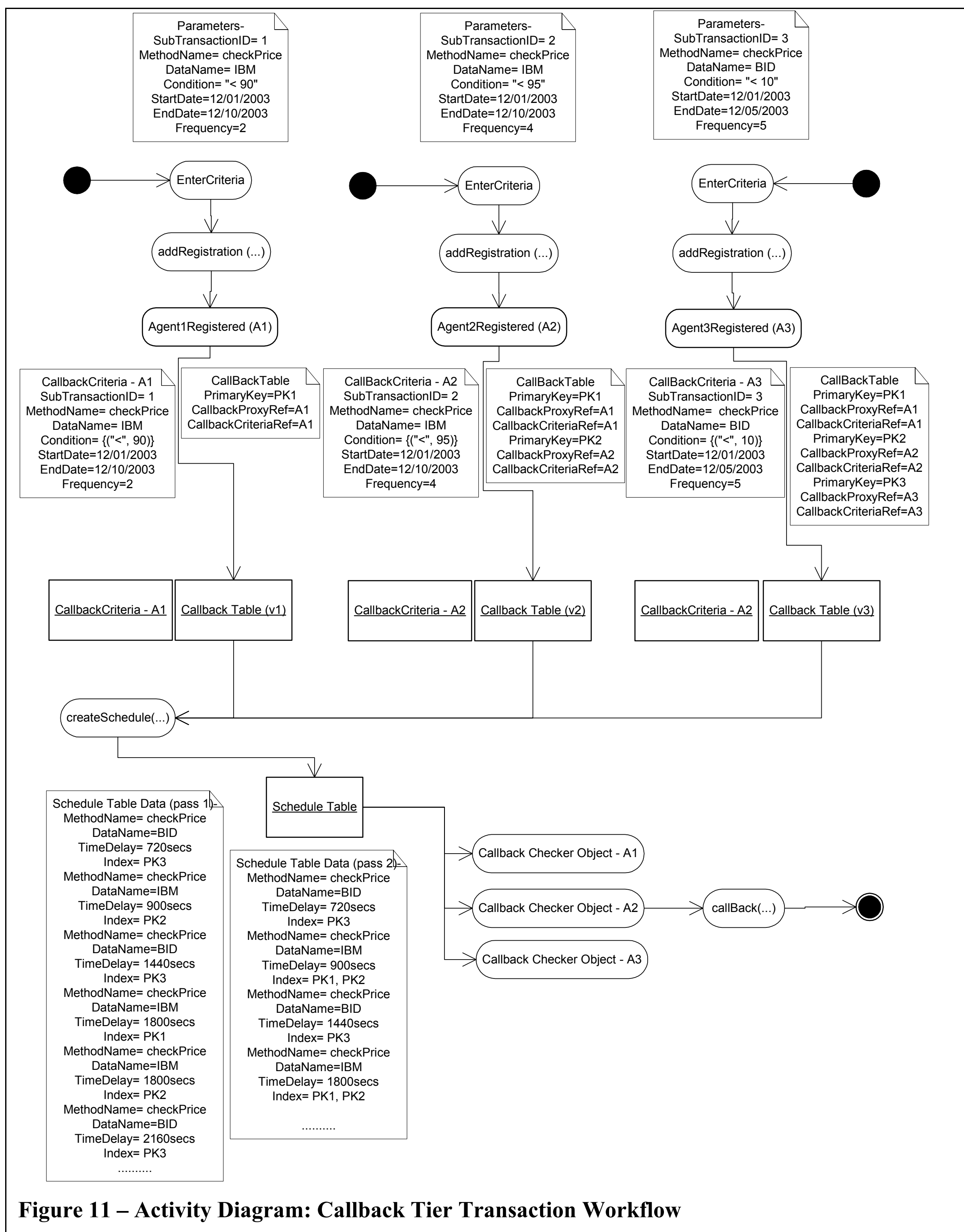

Figure 11 – Activity Diagram: Callback Tier Transaction Workflow

4.8 Summary

This chapter further develops the tiered software architecture strategy used in most contemporary e-commerce portals by creating a new tier to manage server-side callbacks. These callbacks are critical for executing delayed transactions and thus

providing relevant information back to the transaction agent for its evaluation. The major design constructs developed for the Callback tier were:

1. A publicly exposed (via Web services) registration API that enables the transaction agent to establish a callback relationship with the e-commerce portal for delayed transactions. This API is used for the registration functions;
2. A Callback Scheduler function that utilizes the input passed from the registration process and Callback table to create a Schedule table for business logic invocations. The Scheduler creates an efficient schedule by satisfying multiple agents that may need to check the same portal data with a single call;
3. A threaded Callback Checking function that utilizes the Schedule table and invokes the business query methods on the database tier to retrieve the most current data. If a state change has occurred, it checks to determine if it satisfies the trigger condition set at registration, and if so, a standardized Callback method call is made to the client-side Callback service to notify the GMAP to retrieve the updated results.

In summary, the Callback tier provides the design constructs to support the requirement of delayed transactions thus enabling transaction agents to react to state changes in the environment and executing the appropriate transactions with the e-commerce portal.

Chapter 5 – A Generic Client-Side Agent Interface Support System

This chapter describes the details of the client-side Agent Interface Support System. The interface and components to communicate with e-commerce portals are described and defined. Interoperability between the transaction agent and server-side support system is achieved through the Generic Middleware between Agents and Portals (GMAP), which is described with a logical design. Lastly, the workflow with transaction agents is illustrated along with typical use cases associated with this support system.

5.1 Interface Abstraction of Transaction Agents

Because the development of transaction agents is still in an immature state, standards and frameworks are still evolving to aid in the definition and development of agent functionality. This is particularly true with the interface in which the transaction agent interacts with the outside environment. Since the agent can operate in a supervised or unsupervised manner, the interface must support an ability to interact with its environment without human user interaction. Also, for this dissertation, since transaction agents are presupposed to be generic in nature, they must be designed to work with a broad range of portals and not specifically designed for a particular portal. An abstraction of the portal interface that exposes its business logic through cooperative client and server-side support systems can have a significant impact on the functionality to broker the interaction between the generic transaction agent and e-commerce portal.

The specific element that interacts with the transaction agent is the proposed Generic Middleware between Agents and Portals (GMAP). The GMAP insulates the

transaction agent from the details of the interface and internal workings of the services through the use of a data-driven approach (i.e., the transaction agent only interacts with data tables). Several assumptions can be made regarding the component relationships and other factors:

- Both the transaction agent and GMAP relate to each other and the outside world as black boxes, meaning that they only expose an API/tables and no internal design or data assets;
- The transaction agent only interoperates with the GMAP and human user and no other external entity;
- As mentioned earlier, although critically important security is out of scope of the dissertation, so authentication between the components and/or e-commerce portal is not directly addressed;
- That the process is triggered by the human user entering key decision parameters into a table, which defines the criteria for the agent to evaluate.

The GMAP is a major element of the client-side support system and can be deployed as a generic and reusable software component. An alternative of the GMAP is a custom middleware that is specifically designed for a particular category of agent or portal. This specialized middleware provides the specificity to address custom portal requirements, but lacks the generic design components to allow a ubiquitous deployment. This is a significant disadvantage prompting the need for a more flexible and cost-effective design that provides a simpler method of accessing the vast array of different e-commerce portals.

5.2 GMAP: Generic Middleware between Agents and Portals

The Generic Middleware between Agents and Portals (GMAP) utilizes a component and table-based design to implement interoperability between the transaction agent, public UDDI registries and e-commerce portals. The major benefit of GMAP is that it reduces the changes required within the portal and insulates the transaction agent with a data-driven design. GMAP consists of the following functions: *Registry Query Manager, Portal Invocation Manager* and *Callback Web Service*. Also, a run-time *E-Commerce Proxy* is generated as part of the registry query process for each portal of interest. The interaction between these components is modeled in a sequence diagram shown in Figure 12.

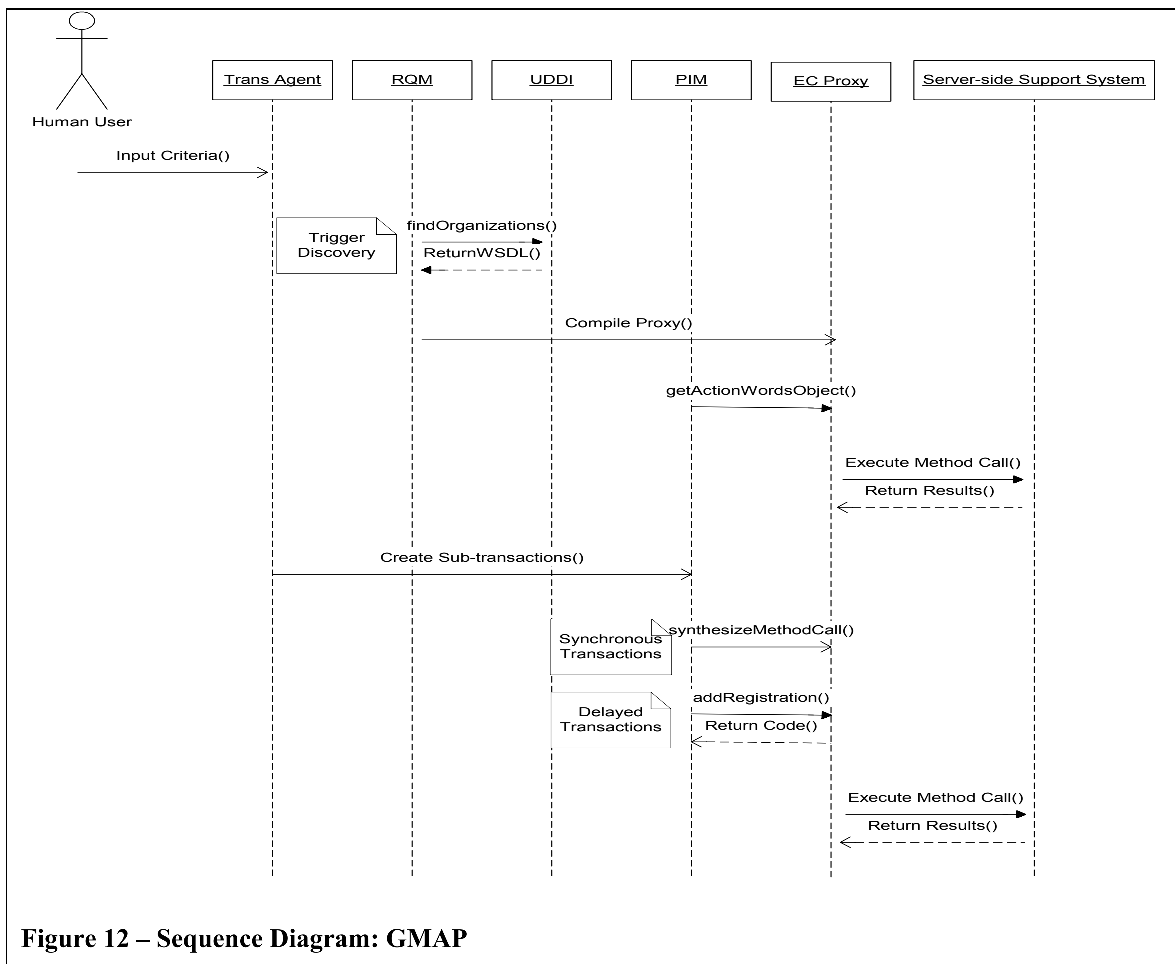

Figure 12 – Sequence Diagram: GMAP

5.2.1 Registry Query Manager (RQM)

The Registry Query Manager is a service that performs an inquiry of a UDDI registry to locate appropriate portals from the business transaction parsed by the transaction agent. The RQM can be implemented as a JAXR client or utilize the UDDI inquiry API (e.g., the findOrganizations or find_business methods will be used depending on whether the JAXR or UDDI inquiry API). The query can be constructed to find the appropriate portals by name, business identifier references, or categories (i.e., genre of business classification). RQM retrieves the appropriate service definitions/bindings (i.e., WSDL) and creates platform-dependent proxy source code and compiles it into executable classes/interfaces that are portal-specific and operate as a proxy for the remote e-commerce portal. The data is stored in the Service Definition table for persistence. The UDDI data model presents a limited number of differentiating data attributes, so the inquiry of the public registry may locate too many portals. This provides an opportunity for the transaction agent to use its logical reasoning capability to filter the interesting portals based on the criteria entered by the human user as part of the business transaction.

5.2.2 Portal Invocation Manager (PIM)

The Portal Invocation Manager (PIM) has four main functions: (1) retrieve the portal-side ActionWords object, (2) map the ActionWord to the MethodName, (3) synthesize the method call and signature and invoke the call via the E-Commerce Proxy (this method call may also be a registration for delayed transactions) and (4) write the results from the response to the *Results table*.

To retrieve the ActionWords object, the PIM reads the Service Definition table to obtain the ProxyReference, makes a call to the server-side Agent Interface Servlet to

retrieve the ActionWords object, which is stored for persistence as the Method Description table. The Method Description table contains a collection of mapped action words to method names for the portal (as well as other attributes explained in section 5.3.5).

The transaction agent initially populates the Sub-transactions table with the individual sub-transactions by parsing and interpreting the input statement entered by the human user. The transaction agent uses the ParameterNames attribute (Method Description table) to build the appropriate ParameterValues attribute (stored in the Sub-transactions table), which is a vector representing the parameters for the method call. The Portal Invocation Manager utilizes the ActionWord that is in the Sub-transactions table to lookup the portal-specific MethodName (from the Method Description table). The PIM uses the MethodName to lookup the method signature in the WSDL and synthesizes the method call (using the ParameterValues as the actual method parameters) for each sub-transaction and passes it to the E-Commerce Proxy (utilizing the appropriate ProxyReference created during the discovery process). The E-Commerce Proxy (section 5.2.4) in turn works with the run-time SOAP processor to create a SOAP request and passes it to the e-commerce portal via an HTTP request. The response to the request is returned to the PIM and the results are written to the Results table (see 5.3.5 and 5.3.6 for further details).

The PIM repeats this workflow and continues to process all sub-transactions associated with the particular business transaction for all portals of interest. Once all sub-transaction results have been returned, the transaction agent evaluates the feedback. Based on this feedback and using its logical reasoning capability, the transaction agent

may create additional sub-transactions in its effort to achieve the business transaction (e.g., maybe the price is the same on portal A and B, but the shipping terms are better on portal B so create additional sub-transactions to execute the purchase on portal B). In the case of delayed transactions, the PIM synthesizes the `addRegistration`, `changeRegistration` or `deleteRegistration` method calls (see section 4.4.1 for details). Also, it instantiates the CallbackCriteria object to be passed as part of the `addRegistration` method call.

5.2.3 Callback Web Service

The Callback Web service presents a standard interface to e-commerce portals that have the Callback tier deployed. The function of the Callback Web service is to notify the PIM of attributes being monitored by the Callback tier (see 5.3.5 and 5.3.6 for further details). A *Client Interface Servlet* is defined in the client WSDL and generated as part of the GMAP deployment, which invokes the method that notifies the PIM to execute a synchronous (pull) method call via the information in the *Callback Registration table* to retrieve the changed data and update the Results table. The Web service runs in a Web server container on the client exposing a single method call to the portal. The interface signature is as follows:

```
void callBack (int subtransactionID);
```

5.2.4 E-Commerce Proxy

As described in section 3.4.5, the E-Commerce Proxy is generated as part of a run-time process that is triggered by the Registry Query Manager. The tool uses the WSDL file as input and creates a portal-dependent class to assist the GMAP in its operation with a portal containing all business API exposed methods along with specialized methods required to execute the actual call via the proxy. Because the E-

Commerce Proxy can not execute a remote method via a text string representing the remote method name and parameter values, a "meta" interface must be created along with the proxy. The Java Class and reflection API (java.lang.Class and java.lang.reflect.Method) can be used to convert a string form of the method name into a dynamic proxy method invocation. Specifically, the Class and reflection API can:

- Create an instance of a class object for the proxy defined in the WSDL at run-time (e.g., `newInstance()`);
- Determine the method declarations that belong to the class object (e.g., `getDeclaredMethods()`);
- Get information about the class modifiers, fields, methods, constructors and superclasses (e.g., `getName()`, `getParameterTypes()`, `getReturnType()`);
- Invoke the underlying method of the proxy with the appropriate arguments at run-time (e.g., `invoke()`).

The remote method call is made via the `invoke` method and passed to the client-side JAX-RPC run-time system. The `invoke` method utilizes two parameters: an array of argument values to be passed to the invoked method and an object whose class declares or inherits the method. The run-time system converts the remote method call into a SOAP message (wrapping all arguments) and transmits the message as an HTTP POST request. The return values are casted to the proper type per the WSDL definition.

5.3 A Logical Design of GMAP

The logical design of the Generic Middleware between Agents and Portals is illustrated in Figure 13. In addition to the major functions, GMAP consists of several tables.

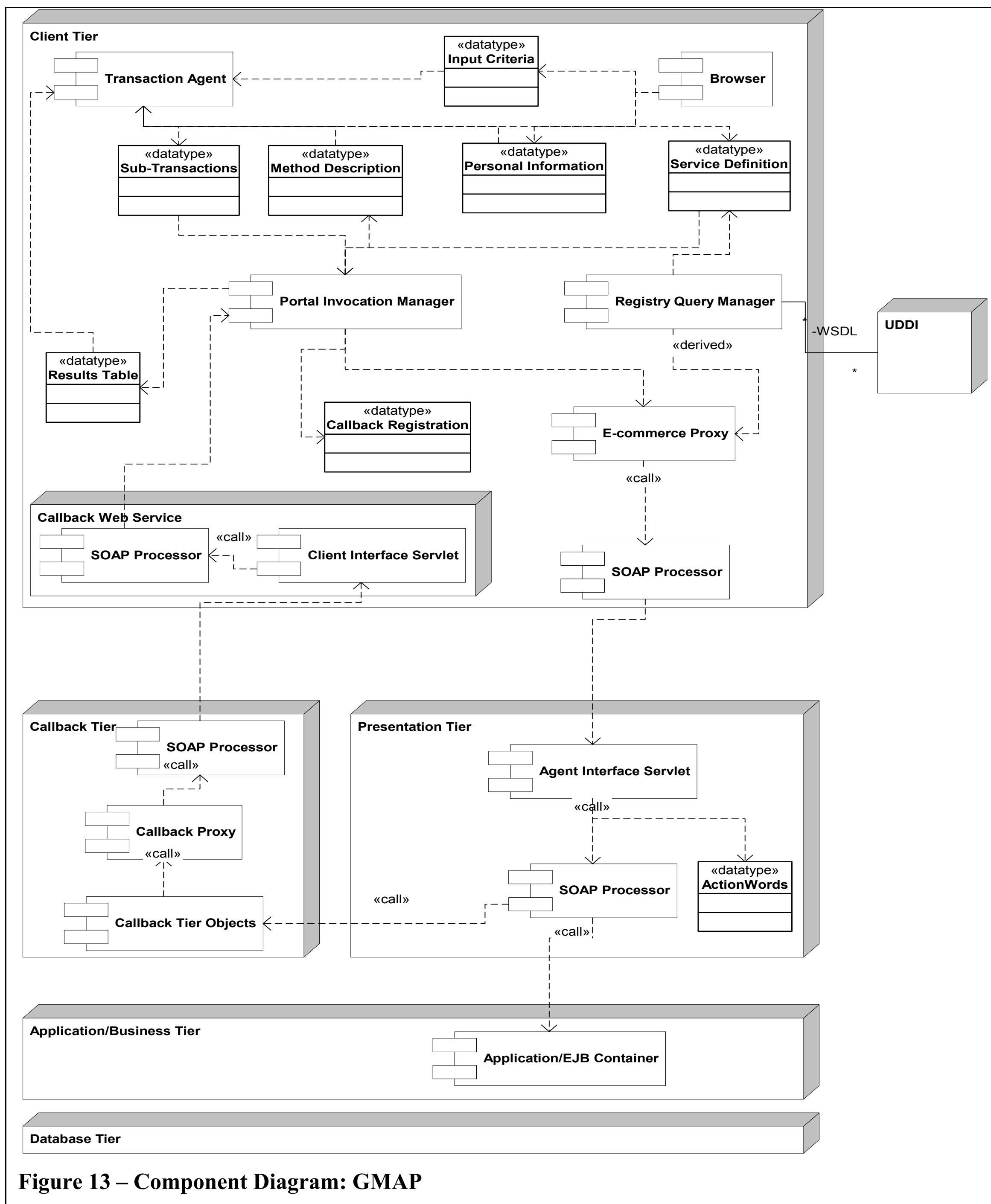

Figure 13 – Component Diagram: GMAP

5.3.1 Input Criteria Table

The Input Criteria table contains the input string entered by the human user representing the business transaction or goal of the human user. The business transaction may be entered as a natural language phrase, which the transaction agent parses and generates sub-transactions from. If the human user modifies the input criteria or the agent receives feedback, additional sub-transactions may be incrementally created over time. The business transaction is in the representative form of satisfying an unmet need (the need identification stage of the CBB model). The attributes maintained by the table include:

Attribute	**Description**
BusinessTransactionID	• A numeric auto incremented unique identifier used to identify the "logical" business transaction (which will be converted into a series of sub-transactions).
RequestString	• A natural language statement representing the unmet purchase need of the human user transaction (e.g., "Buy the XYZ book for less than $30, if it can be delivered in the next 5 days.").

Table 6 – Input Criteria Table

5.3.2 Sub-transactions Table

The transaction agent uses the standardized action words to reflect the semantic intent of each of the sub-transactions. Also, as mentioned earlier regarding input criteria, additional sub-transactions may be created incrementally as the transaction agent processes additional input from the human user or as a reaction to feedback from previous method calls. If the input statement specifies a duration period for the business transaction, then the transaction agent populates the StartDate, EndDate and Frequency.

PIM treats these as delayed transactions (push mode). Otherwise, the sub-transactions are processed as synchronous method calls (pull mode).

For each row in the table is a SubTransactionID, which is a unique identifier that is auto incremented for each sub-transaction created by the transaction agent. If the information entered by the human user is insufficient to generate a particular method call for a portal, an error message is generated for the human user to adjust the input criteria. Rows are deleted from the table by the PIM when the business transaction completes (successfully or unsuccessfully for synchronous transactions) or after the end date for a delayed transaction or the human user cancels a callback registration. The Method Call table attributes are:

Attribute	Description
SubTransactionID	• A numeric is auto incremented and stored as a unique identifier for this particular agent sub-transaction. In addition, for delayed transactions, it is passed as part of the callback registration process and passed back for transaction identification when a callback occurs.
BusinessTransactionID	• The attribute (described in the Input Criteria table) is stored as a foreign key reference to the original "logical" business transaction.
ActionWord	• Key word actions to be converted into method calls by the PIM. For example, the action may represent a price check (CheckPrice). The action words are used to lookup method names, parameter and return value characteristics from the Method Description table previously retrieved from the portal.
ParameterValue(s)	• A vector that represents the actual argument values for the particular method signature for which the ActionWord maps to. The parameter values are parsed from the business transaction statement by the transaction agent. For example, the value(s) may represent the identifier for the item of interest, i.e., a book ISBN.

Table 7 – Sub-Transactions Table

5.3.3 Callback Registration Table

For delayed transactions, the Portal Invocation Manager registers the agent with the e-commerce portal using the exposed `addRegistration` method (refer to section 4.4.1 for details). It also maintains a record of the registration in the GMAP in the Callback Registration table since the agent may be registered on multiple portals for callback. The PIM utilizes information in the Sub-transactions table, Method Description table object and the WSDL to construct the signature for the registration method call. The SubTransactionID attribute is the unique identifier for a particular method call, which is passed along with several other parameters during registration and is passed back in the callback to trigger a pull method call to retrieve the updated data.

Also, a string object is created holding the WSDL file for the client-side Callback Web service, which is passed during callback registration. It also constructs the condition statement (see section 4.4.1 for details) from the attributes stored in the Input Criteria table, as well as the duration (start and end dates/times) and checking frequency for the delayed transaction period. These attributes are written to a CallbackCriteria object, which is passed as a serialized object. Rows are deleted from the table when the callback period is exceeded or a `deleteRegistration` call is made to the portal. The Callback Registration table attributes are as follows:

Attribute	**Description**
SubTransactionID	• A unique identifier created and stored on the client representing the sub-transaction of the method call for the attribute being monitored. It is also passed as part of the registration process and passed back if a Callback occurs.
MethodName	• The name of the method on the portal.
ParameterValue(s)	• Same definition as the Sub-transactions table. This is passed during callback registration as the

	dataName parameter.
Condition	• A formatted relational statement passed as part of the registration process to the server. It is used by the server-side support system to determine if a callback should be made to the client.
StartDate	• Date object representing the start date/time of the callback period.
EndDate	• Date object representing the end date/time of the callback period.
Frequency	• Frequency of database queries per hour.
ServerCallbackID	• A unique identifier created and returned by the `addRegistration` method call. This ID is passed back to the portal to identify the correct registration for the `changeRegistration` and `deleteRegistration` method calls.

Table 8 – Callback Registration Table

5.3.4 Service Definition Table

The Service Definition table is updated during the registry discovery phase by the Registry Query Manager. The ProxyReference is created when the RQM creates and validates the proxies at registry discovery time. This ProxyReference is used by the PIM to direct the method calls to the appropriate proxy. It has one row per each registered e-commerce portal. The table contains the following attributes:

Attribute	**Description**
UUID	• Universal unique identifier of the business (i.e., portal) created by the registry
BusinessTransactionID	• The attribute (described in the Input Criteria table) is stored as a foreign key reference to the business transaction.
Name	• Name of business
Description	• Text that describes the business
ContactInformation	• Contact name, mailing address, telephone and email address
Classification	• 3rd party classification codes representing the type of business/industry
WSDLReference	• Location reference (e.g., URL) of portal WSDL

ProxyReference	• A reference to the E-Commerce Proxy generated during the discovery phase.

Table 9 – Service Definition table

5.3.5 Method Description Table

The Method Description table is populated from the portal ActionWords object and is used to develop a semantic understanding between a generic transaction agent and portal that enables the agent (via the PIM) to invoke the proper business logic method, and correctly interpret the parameter and return value list. This object is resident on the portal and retrieved by the client-side support system as part of the portal invocation cycle. The object contains the following attributes:

Attribute	**Description**
ActionWord	• Action words representing key words for method names or parameters/return values used by the PIM and transaction agent.
MethodName	• The name of the method on the e-commerce portal. For example, ChkPrice may be the actual method name for checking a price (as mentioned, the action word is CheckPrice).
ParameterNames	• A vector of strings that represent the action words for each of the parameters for the method signature of the MethodName. The order of these action word vectors is consistent with the method signature defined in the WSDL.
ReturnValueNames	• A vector of strings (same construct as ParameterNames) representing the action words of the return value(s) for the method call. The order and data type is defined in the WSDL. These action words are used by the transaction agent for interpreting the semantics of the method call response.
ReturnValueStructure	• A vector of strings representing the object structure of the return values matching the data types in the WSDL. The structure of the data members is defined using the dot notation convention.

Table 10 – Method Description table

The ParameterNames attribute (along with ReturnValueNames) are used to span the semantic gap problem that exists between the transaction agent and external environment. For example, a portal method for checking item pricing may have two parameters, a unique item identifier and a marketing promotion code. The first vector position for ParameterNames containing string values may be defined with several common action words used by the transaction agent for semantic interpretation (e.g., ["Item ID", "ISBN", "Product Code", …]), while the second vector position may be represented by (["Promotion Code", "coupon", …]). ParameterNames are used as key words for the transaction agent's logical reasoning capability to match criteria entered by the human user in order to create the sub-transactions, which contain the actual parameters (i.e., ParameterValues attribute).

The ReturnValueNames and ReturnValueStructure attributes are used in combination to support flexibility of return values that may exist in the business API of the portal. Since method call responses are not constrained in this solution, the support system provides design constructs and has common functions to decompose the return value structure into its atomic data elements, which can be written to a columnar table format. To support a decomposition of the return results into columnar data, it is assumed that in their atomic form, all data members are represented by character or primitive data types. For example, a portal method for checking item pricing may return three values, the unique item identifier, a currency code and the price. For each ReturnValueNames vector position there would be a series of actions words representing the meaning of the return value (e.g., ["Product Code", "Item Code", ...], ["currency code", "ISO code", …], ["Price", "Amount", "Cost", …]). These return values may be

represented by more complex data structures and encapsulated objects on the portal. The pseudo-code example for such a structure is as follows:

```
class ItemData {
        String ItemCode;
        PriceData pData;
                }
class PriceData {
        String currency;
        double price;
                }
```

In this case, the corresponding ReturnValueStructure attribute is:

["ItemCode"], ["pData.currency"], ["pData.price"]

The ReturnValueNames and ReturnValueStructure attributes are used by the PIM to decompose the return value into its individual data members, cast those data members into their appropriate return data type (as defined in the WSDL) and write the casted data members to a columnar Results table.

5.3.6 Results Table

The Results table is updated by the results returned to the GMAP from the synchronous method calls. The synchronous method call response from the JAX-RPC call updates the table via the Portal Invocation Manager. The delayed transactions response is via the standardized callBack API returning the SubTransactionID to notify the PIM to invoke the proper business logic method (by referring to the Callback Registration table) to retrieve the latest portal datum that changed (using the established pull/synchronous mechanism). The table contains the following attributes:

Attribute	**Description**
SubTransactionID	• The unique identifier assigned to every sub-transaction made either as a synchronous call or delayed transaction.
ResultsValueColumn1…XX	• A series of columns holding the character

	or primitive data type return values written from the synchronous or delayed transaction calls.

Table 11 – Results Table

5.4 Integration of Transaction Agents with GMAP

To have an effective integration of generic transaction agents with GMAP, it must be a reusable software module easily deployable across portals and compatible with the server-side support system. The client-side deployment tool creates the base service modules, tables and Web service components (e.g., SOAP processor) for the GMAP. The GMAP design is made generic by the following approaches:

- Portal specific E-Commerce Proxies (or stub classes) are compiled at run-time representing the exposed Web services interface for transaction agent support system to use;
- The internal workings and complexity of a transaction agent are insulated from the GMAP by the data-driven table-based design;
- An external registry discovery process adjusts the specific configuration of the GMAP using elements retrieved from the registry and the portals (i.e., WSDL and ActionWords object) to make it compatible with all registered e-commerce portals;
- It implements a standard Callback Web services to support delayed transactions. Since this interface is standardized, it will be available to all portal environments with the plug compatible server-side support system implemented;
- The semantic understanding between the human user/transaction agent and portal is achieved by mapping action words that represent actions within the

CBB model to the specific method calls, parameter and return values of the e-commerce portal.

In order for a transaction agent and client environment to utilize the generic GMAP, it must comply with several guidelines and standards to ensure integration with the server-side support system. Those integration strategies are:

- The transaction agent has the design characteristics as described in section 3.1.3 and generally complies with other functional aspects defined in section 5.1;
- The client environment must have the infrastructure (i.e., memory, storage, etc.) to support the deployment of the GMAP;
- The client must operate from a fixed IP address to support callback for delayed transactions;
- The client must execute a Web container to expose the Callback API.

Likewise, in order for the portal to work with GMAP and expose its selected business logic to the remote transaction agents, the following major steps should be taken:

- The business logic methods to be exposed to the public remote transaction agents need to be identified. Normally these are the same EJB methods invoked from the presentation tier by human users with Web browsers;
- With the signatures of these selected business logic methods as input, a generic Web service tool will need to generate a WSDL 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;

- The WSDL file needs to be registered with public UDDI registries under proper industry categories;
- Implement a portal-specific ActionWords object, which allows the portal designer to associate the action words, standardized for transaction agents to map generic transaction operations to business logic methods. The object will also provide keyword descriptions for all method signatures, specifically the parameter and result values and a getter method for each of the value components of a returned object;
- The entry point Agent Interface Servlet is deployed in the presentation tier servlet container, which will expose the selected business logic methods for transaction agents to access.

5.5 Typical Use Cases and Workflow: Pull Mode

The primary use cases revolve around the purchase objective of the human user. In a purchase scenario example, a human user leveraging a transaction software agent wants to buy an item from one of a number of different e-commerce portals. The main actors are the human user (i.e., buyer) and transaction agent. The human user is the main initiator who triggers the transaction agent's interaction with the e-commerce portal by establishing the decision-making criteria used by the agent in executing the business transaction.

The transaction agent initiates the find merchant use case to query the public registry (yellow pages) for e-commerce portals that can satisfy the unmet product/service

need. The product search use case relates to the CBB product brokering phase in which information (e.g., price, inventory availability, etc.) is retrieved resulting in a consideration set (stored in the Results table) for the transaction agent to evaluate. The next CBB phase (merchant brokering) results in the agents utilizing this information to choose the merchant to transact business with. If price isn't set (e.g., auctions), then negotiation use cases may be applicable (e.g., bidding), otherwise the last phase of the CBB model (purchase and delivery) occurs. In this phase the order is processed, payment is made, and delivery options chosen.

As an example to demonstrate how the proposed GMAP design will operate, a series of workflow steps associated with the use cases and activity diagram are documented in Figure 14. The primary scenario is a non-auction purchase event, which can be achieved by using a subset of e-commerce related use cases: FindMerchant, SearchProductCatalog, BrowseProduct, and CreateOrder (which uses the UpdateOrder and ProcessOrder use cases). BrowseProduct uses additional use cases, specifically in this scenario CheckPrice and CheckAvail.

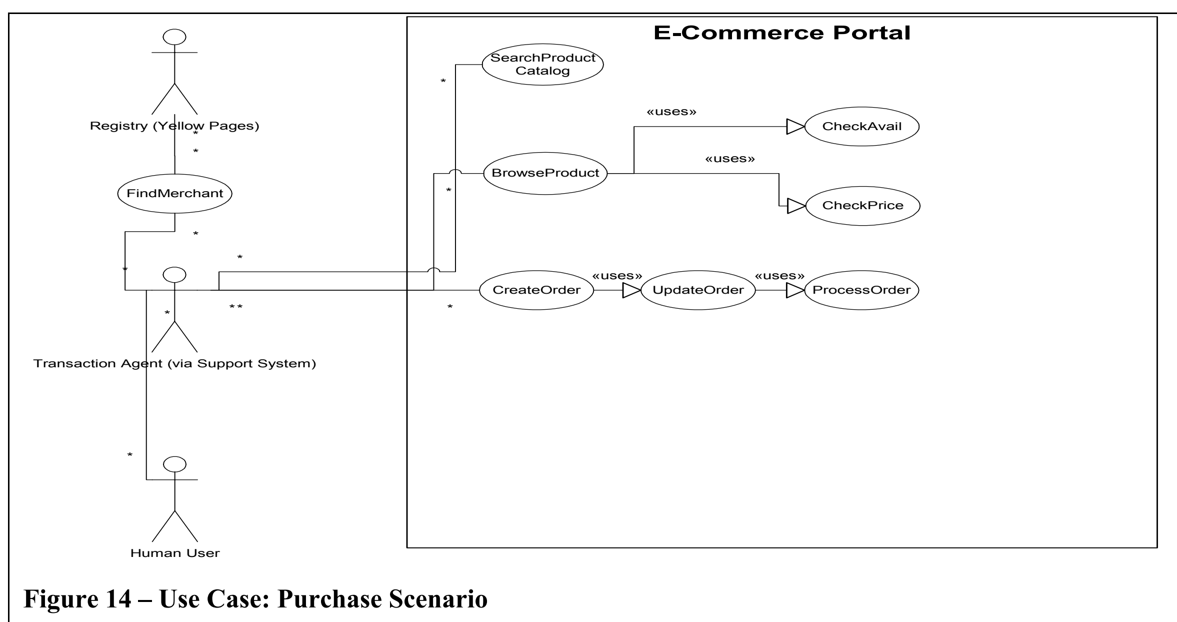

Figure 14 – Use Case: Purchase Scenario

To illustrate the concept, the workflow for a purchase event will be described in terms of the GMAP and illustrated in an activity diagram. The workflow steps associated with this use case are illustrated in Table 12 (some details were described in earlier sections).

Workflow
1. The human user establishes the business transaction, which satisfies an unmet need for purchasing a particular book by entering criteria consisting of an item identifier (such as the ISBN, manufactures number or a description) and maximum price to pay. Other parameters may be required for a complete interaction, but these are key parameters needed to achieve the purchase objective.
2. The transaction agent parses the business transaction into actions (using action words) and parameters (using the ParameterNames attributes) used by the GMAP services for synthesizing method calls and filtering.
3. The transaction agent creates an initial series of sub-transactions for each of the established portals, which are stored in the Sub-transactions table. The table contains a set of rows that when complete represent one logical transaction (e.g., purchase a book). These sub-transactions represent the required actions for a sub-set of the use cases for each of the portals (SearchProductCatalog, CheckPrice and CheckAvail).
4. This triggers the Registry Query Manager function to query (using the parsed key words) the public "yellow pages" registry (FindMerchant use case) and at run-time, create a proxy for each e-commerce portal.
5. The Portal Invocation Manager uses the ProxyReference created by the RQM to retrieve the correct ActionWords object from the correct portal. The data (action words) in the ActionWords object is stored in the Method Description table.
6. The Portal Invocation Manager processes each of these Sub-transaction table rows by using the ActionWord (which was determined by the transaction agent) to look up the method name in the Method Description table, and then matching it with the method name in the portal WSDL to synthesize a method call with the appropriate method signature. It uses the parameters stored in the Sub-transactions table created by the transaction agent.
7. This method call is passed to the local E-Commerce Proxy, which is the same method as the remote business logic method, wraps the invocation in a SOAP (XML) message and sends it to the portal's entry point with an HTTP request.

8. 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.
9. The E-Commerce Proxy converts the response SOAP message into the actual response data and type. The Portal Invocation Manager decomposes the response data into its primitive form (using the ReturnValueStructure attribute) and writes the results to the Results table (see sections 5.3.5 and 5.3.6).
10. Once all the responses for each of the portals are returned for the method calls represented by the SearchProductCatalog, CheckPrice and CheckAvail use cases, the transaction agent uses its logical reasoning capability and evaluates the results for the “best” option (given price and inventory availability were the criteria entered by the human user).
11. Once the evaluation is done and a decision is made, the transaction agent uses its logical reasoning capability and generates incremental sub-transaction for the remaining use cases (CreateOrder, UpdateOrder and ProcessOrder) following steps 6-9. If no option met the criteria, the process ends with an unsuccessful purchase event.

Table 12 – GMAP Workflow

The activity diagram matching these use cases and workflow is highlighted in Figure 15.

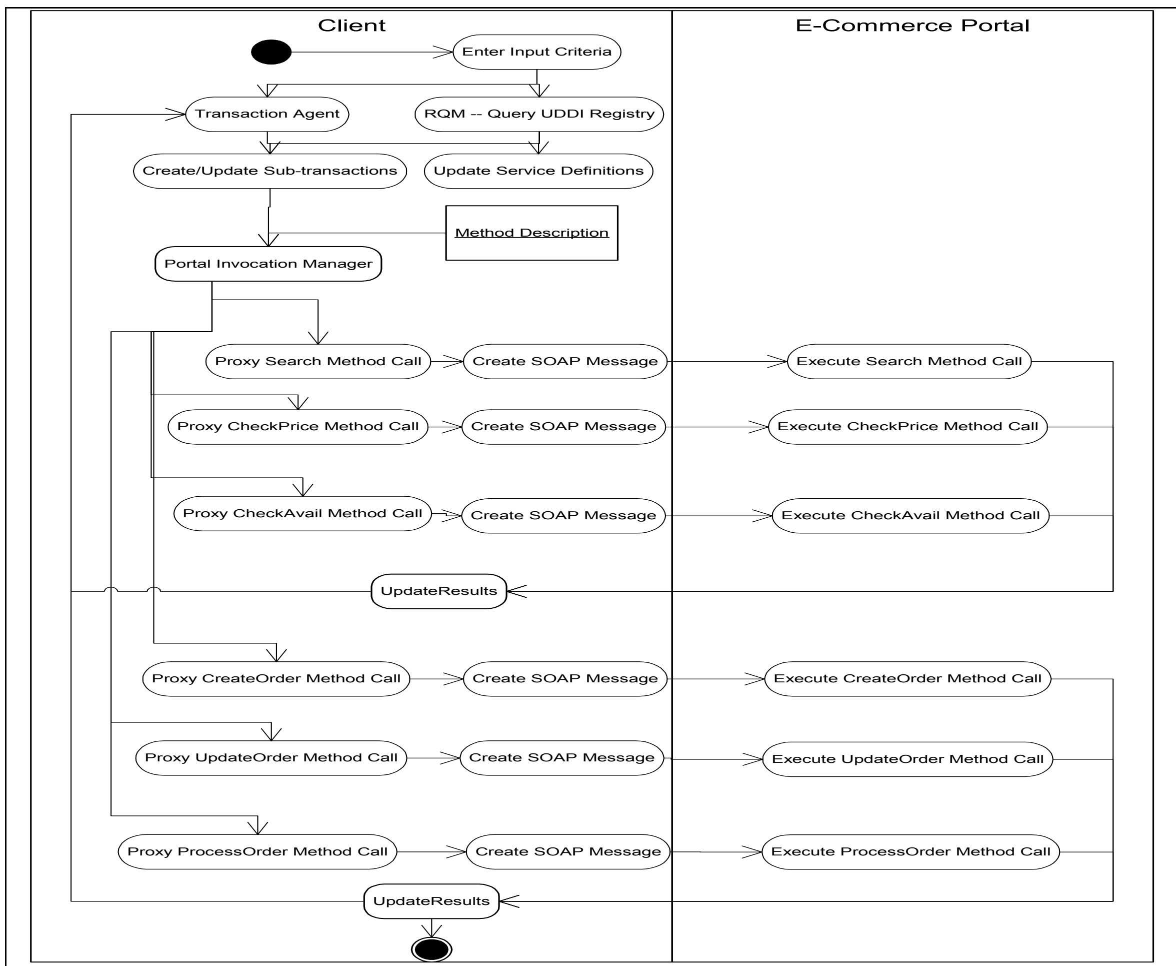

Figure 15 – Activity Diagram: GMAP Workflow

5.6 Summary

This chapter describes the design of the client-side support system that works in conjunction with the server-side support system. The main element of the client-side support system is the Generic Middleware between Agents and Portals (GMAP). The major design constructs developed for the GMAP were:

- Registry Query Manager is a service that searches a UDDI registry for e-commerce Web services. The WSDL and other registry information is retrieved and stored in the Service Definition table. It also creates the

individual E-commerce Proxies at run-time, which expose the business API of the portal via Web services method calls.

- Portal Invocation Manager is a service responsible for processing the rows in the Sub-transactions table, which represent the steps required to accomplish the business or logical transaction. The transaction agent populates the Sub-transactions table with the individual sub-transactions by parsing and interpreting the input statement entered by the human user. The PIM then retrieves the portal-side ActionWords object, maps the ActionWord to the MethodName, synthesizes the method call and signature using the WSDL, invokes the call via the E-Commerce Proxy and writes the results from the response to the Results table.
- Callback Web service is a Web container running a Client Interface Servlet listening for callback calls from the e-commerce portal. It presents a standard interface to e-commerce portals that have the server-side support system implemented. The function of the Callback Web service is to update the Results table with any updated data from attributes being monitored by the e-commerce portal.

In summary, the GMAP is a plug compatible component that interoperates with the corresponding portal support system. It provides the design constructs to support transaction agents that conform to generalized design specifications, thus minimizing modifications to them by utilizing e-commerce portal information from public registries and abstracting the interface requirements into the GMAP, which lessens the burden on the transaction agent.

Chapter 6 – Conclusion

The final chapter outlines the major contributions made from this research and highlights opportunities for future research into this problem.

6.1 Major Contributions

To-date much of the research has focused on the agent and the capabilities of the agent. The solution strategy described in this research provides an alternative viewpoint in which the focus is on the environment the agent operates within. This research focused on architectural solutions to agent-enabling e-commerce portals with pull/push abilities. The design solutions improve interoperability between generic transaction agents and e-commerce portals, and thus enhance their ability to execute back-end business transactions, while maintaining the traditional graphical user interface for the human user. The major contributions include:

1. An Agent-Enabling Interface that exposes an open business API. This is achieved through a cooperative relationship between a client and server-side Agent Interface Support Systems that leverages Web services. The two support systems in effect act as middleware between the transaction agent and the e-commerce portal;
2. A generic Callback tier to support delayed transactions that enables transaction agents to react to changes of state within in a timely and efficient fashion (a feature not natively supported with Web services);
3. A targeted solution for creating semantic meaning between the transaction agent and e-commerce portal through the use of an action words based

translation system that bridges the semantic gap by codifying the method name, parameter and return value names to action words;

4. A Generic Middleware between Agents and Portals (GMAP) that enhances interoperability between transaction agents and e-commerce portals by managing the portal discovery process through UDDI, retrieving pertinent information about the portal's specific interface and establishing E-commerce Proxies.

The design solutions targeted the environment in which the transaction agents operate within, specifically the technology and architecture deployed in existing e-commerce portals. To-date, much of the research in this area focused on the agent. Multi-Agent Systems (MAS) operating within a standardized society of agents, agent communication languages, autonomous and mobile agent behavior models represent just a few of the strategies or techniques prevalent in the existing literature. By targeting the operating environment with support system enhancements, the approach defined in this research provides for several benefits:

- It solves a difficult problem associated with mobility. It does this by bringing the e-commerce portal to the client (in the form of the E-Commerce Proxy) versus bringing the transaction agent to the e-commerce portal. Mobility requires transfer of the agent by suspending it action, identifying the agent state to transfer, serializing the agent class and state and transferring the serialized version of the agent. Likewise, on the receiving side (service-side), the agent must be de-serialized, instantiated and execution resumed. These are complex actions requiring elaborate design constructs for agent life cycle

management, transfer and naming facilities across multiple hosts and are mitigated by the use of client-side E-commerce Proxies.

- No Agent Communication Language (e.g., KQML or FIPA ACL) is required for agent communication since the design directs the dialog between the transaction agent and the e-commerce portal by leveraging the existing business logic API of the portal.
- Since the existing business API of the e-commerce portal is exposed via the support systems utilizing Web services, the transaction agent doesn't need to utilize screen-scraping techniques thus enabling a generic transaction agent design.
- The solution maintains the typical request/response workflow, which is the foundation of most e-commerce portals by utilizing the HTTP GET/POST protocol and executing the same back-end components as the human user (the human user accesses back-end components via a JSP). In essence, the application workflow can remain the same regardless whether the portal is servicing the needs of a human user or transaction agent.
- Since the solution operates as a typical request/response workflow, the e-commence host environment doesn't need to provide for containment to secure the host environment from alteration by the mobile transaction agent. The transaction agent is not freely executing on the host environment, but rather is bounded by the methods made available to access the back-end components.

- The transaction agent is data driven meaning it only requires interface components capable of reading and writing to tables.
- Since the Agent Interface Support Systems are installable and compatible reusable components, they provide for support of existing e-commerce portals requiring less change of the portal's architecture and eliminates change of the portal back-end.

6.2 Future Work

This research also highlights additional challenges or other areas of unresolved issues. For example, a significant area of research regarding the Semantic Web could greatly improve agent-portal communications through the use of XML, Resource Description Framework (RDF), and domain-specific (i.e., e-commerce) ontologies, which are powerful tools for developing a structured collection of information and inference rules that can assist the agent's logical reasoning capability. Also, the solution was purposely constrained on the types of conditions that would trigger a callback. Additional functionality could be added to this feature of the design solution to provide more flexibility in terms of the criteria and number of attributes monitored. Lastly, security was not dealt with in this research, but it is a mission-critical aspect of e-commerce, so there remain significant challenges in ensuring a highly secure environment for transaction agents to operate within.

References

[1] AgentBuilder, "AgentBuilder User Guide", URL: http://www.agentbuilder.com/index.html, [June 2002].

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

[3] Brenner, W., Zarnekow, R., Wittig, H., "Intelligent software agents: foundations and applications", *Springer*, Germany, ISBN 3540634118, 1998.

[4] Brown, A., Johnston, S., Kelly, K., "Using Service-Oriented Architecture and Component-Based Development to Build Web Service Applications", *Rational Software,* URL: http://www.rational.com/media/whitepapers/TP032.pdf?SMSESSION=NO, [June 2003].

[5] Department of Commerce, "The United States Department of Commerce News", Washington, D.C., URL: http://www.census.gov/mrts/www/current.html, [May 2003].

[6] Feldman, S., Yu, E., "Intelligent Agents: A Primer", *Searcher*, Volume 7, No. 9, October 1999**,** URL: http://www.infotoday.com/searcher/oct99/feldman+yu.htm, [June 2002].

[7] Finin, T., Fritzson, R., McKay, D., McEntire, R., "KQML as an Agent Communication Language", *Proceedings of the Third International Conference on Information and Knowledge Management*, November, 1994.

[8] Finin, T., Weber, J., Wiederhold, G., Genesereth, M., Fritzson, R., McGuire, J., Shapiro, S., McKay, D., Pelavin, R., Beck, C., "Specification of the KQML Agent Communication Language plus example agent policies and architectures", *The DARPA Knowledge Sharing Initiative External Interfaces Working Group*, University of Toronto, 1994.

[9] FIPA, "FIPA Agent Management Specification", *Foundation for Intelligent Physical Agents*, URL: http://www.fipa.org/repository/managementspecs..html, [December 2002].

[10] FIPA, "FIPA Web Site", *Foundation for Intelligent Physical Agents*, URL: http://www.fipa.org , [May 2003].

[11] Franklin, S., Graesser, A., "Is it an Agent or just a Program? A Taxonomy for Autonomous Agents", *Proceedings of the Third International Workshop on Agent Theories*, pp 21—35, 1996, URL: http://www.msci.memphis.edu/~franklin/AgentProg.html, [November 2001].

[12] Gini, M., “Agents and other Intelligent Software for E-Commerce”, *CSOM*, University of Minnesota, February 1999, URL: http://www-users.cs.umn.edu/~gini/csom.html, [June 2003].

[13] Gottschalk, K., Graham, S., Kreger, H., Snell, J., “Introduction to Web services architecture”, IBM Systems Journal, Vol. 41, No. 2, pg. 170-177, 2002.

[14] Grant, E., “E-Commerce To Top $1 Trillion in 2002”, E-Commerce Times, February 13, 2002, URL: http://www.ecommercetimes.com/perl/story/16314.html [January 2003]

[15] Greenwald, A., Kephart, J., Hanson, J., “Dynamic Pricing by Software Agents”, *Computer Networks*, 32(6), pg. 731--752, 2000, URL: http://www.cs.brown.edu/people/amygreen/papers/rudin.pdf, [June 2002].

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

[17] Honavar, V., “Intelligent Agents and Multi Agent Systems”, *IEEE CEC*, Washington D.C., July 1999.

[18] Hughes, M., Shoffner, M., Hamner, D., Bellur, U., “Java Network Programming”, Manning, 2nd Edition, Chapter 25, ISBN 188477749X, 1999.

[19] The Java Web Services Tutorial for JWSDP v1.3, “Java Web Services Developer Pack”, *Sun Microsystems*, July 25, 2003, URL: http://java.sun.com/Webservices/tutorial.html, [October 2003].

[20] Krulwich, B., “The BargainFinder Agent: Comparison price shopping on the Internet”, Bots and Other Internet Beasties, *SAMS.NET*, URL: http://bf.cstar.ac.com/bf/, 1996, [December 2002].

[21] Maes, P., Chavez, A., “Kasbah: An agent marketplace for buying and selling goods”, *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*, London, UK, Practical Application Company, pg. 75—90, 1996.

[22] Maes, P., Guttman, R., Moukas, A., “Agents That Buy and Sell”, *Communications of the ACM*, Vol. 42, No. 3, pg. 81-91, March 1999.

[23] Martin, D., Cheyer, A., Moran, D., “The Open Agent Architecture: A Framework for Building Distributed Software Systems”, Applied Artificial Intelligence, Vol. 13, pg. 91—128, January-March 1999.

[24] Natis, Y., Schulte, R., "Introduction to Service-Oriented Architecture", *Gartner Group*, Research Note, SPA-19-5971, April 14, 2003.

[25] Natis, Y., Thompson, J., "What Can Web Services Do for You?", *Gartner Group*, Research Note, COM-13-8485, July 2, 2001.

[26] Newcomer, E., "Understanding Web Services", *Addison-Wesley,* ISBN 02017508133, 2002.

[27] Nwana, H., "Software Agents: An Overview", *Knowledge Engineering Review*, Vol. 11, No 3, pg. 1—40, September 1996, URL: http://www.sce.carleton.ca/netmanage/docs/AgentsOverview/ao.html[June 2002].

[28] O'Brien, P.D., Nicol, R.C., "FIPA – towards a standard for software agents", *BT Technol J*, Vol 16, No 3, pg. 51--59, July 1998.

[29] OMG, "CORBA Basics", *Object Management Group*, OMG FAQ, URL: http://www.omg.org/gettingstarted/corbafaq.htm, [August 2003].

[30] OMG, "Mobile Agent System Interoperability Facilities Specification (MASIF)", *Object Management Group*, OMG TC Document orbos/97-10-05, November 1997, URL: http://www.hpl.hp.com/personal/Dejan_Milojicic/ma4.pdf, [January 2003].

[31] Raisamo, R., "Introduction: What are the agents?", University of Tampere, URL: http://www.cs.uta.fi/kurssit/agents/agents-lecture3.ppt, [December 2002].

[32] Ramamohanarao, K., Bailey, J., "Transaction Oriented Computational Models for Multi-Agent Systems", *13th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'01),* Dallas, Texas, pg. 11—17, Nov. 7-9, 2001.

[33] Smith, D., "Management Update: A Common Sense Definition of Web services", *Gartner Group*, IGG-06122002-04, Article, June 12, 2002.

[34] Tao, L., "Shifting Paradigms with the Application Service Provider Model", *Computer*, *IEEE*, 2001, URL: http://csis.pace.edu/~lixin/pdfDownload, [November 2003].

[35] Universal Description, Discovery and Integration of Web Services, "UDDIv2 Specifications", *UDDI.org OASIS Technical Committee*, URL: http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv2, [November 2003].

[36] University of Maryland, "Knowledge Interchange Format (KIF)", *UMBC Lab for Advanced Information Technology*, URL: http://www.cs.umbc.edu/kse/kif/, [December 2002].