A Dynamic Workflow Framework for Mass Customization Using Web Service and Autonomous Agent Technologies

Daniel J. Karpowitz
Brigham Young University - Provo

A DYNAMIC WORKFLOW FRAMEWORK FOR MASS CUSTOMIZATION USING WEB SERVICE AND AUTONOMOUS AGENT TECHNOLOGIES

by

Daniel J. Karpowitz

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Mechanical Engineering

Brigham Young University

December 2006

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Daniel J. Karpowitz

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date	Jordan J. Cox, Chair
Date	Jeffrey C. Humpherys
Date	Timothy W. McLain
Date	Sean C. Warnick

ABSTRACT

A DYNAMIC WORKFLOW FRAMEWORK FOR MASS CUSTOMIZATION USING WEB SERVICE AND AUTONOMOUS AGENT TECHNOLOGIES

Daniel J. Karpowitz

Department of Mechanical Engineering

Master of Science

Custom software development and maintenance is one of the key expenses associated with developing automated systems for mass customization. This paper presents a method for reducing the risk associated with this expense by developing a flexible environment for determining and executing dynamic workflow paths. Strategies for developing an autonomous agent-based framework and for identifying and creating web services for specific process tasks are presented. The proposed methods are outlined in two different case studies to illustrate the approach for both a generic process with complex workflow paths and a more specific sequential engineering process.

ACKNOWLEDGMENTS

I absolutely owe much thanks to all who supported and assisted my work: Dr. Jordan Cox for his patience with my complete inability to work to a fixed schedule; my graduate committee members, Dr. Tim McLain, Dr. Jeff Humpherys, and Dr. Sean Warnick, for their insight, direction, and general support of my work; my loving wife, Abby, for her willingness to endure many long and late nights; and my beautiful daughter, Sophie, who always helps put life in context.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1 INTRODUCTION

Tseng and Jiao define mass customization as “producing goods and services to meet individual customer’s needs with near mass production efficiency” [1]. Automation of engineering product development processes is essential to the implementation of mass customization. The objective of mass customization is to provide individualized products and services by integrating processes that are agile and flexible [2]. Production systems that utilize automated, reusable processes are becoming an increasingly important tool for improving the efficiency, accuracy, and cost during the product lifecycle as well as enabling mass customization.

Over the past two decades the development of process automation tools such as Computer Aided Design (CAD), Computer Aided Engineering (CAE), Computer Aided Manufacturing (CAM), Product Data Management (PDM), Product Lifecycle Management (PLM), and Enterprise Resource Planning (ERP) have had a significant impact on process automation. To realize many of the benefits of advanced CAx tools, company processes must be captured and automated through additional software development. The custom software development that is often required to extend and integrate these tools represents a significant investment of both time and money. The risk of such an investment often prevents businesses from implementing mass customization.

The key expense in this custom software development comes from the automation system not being flexible enough to adapt to changes in process tools and product markets. Small changes in the tools, market, or most importantly the product development process results in large additional investments in software modifications. The volatility of product markets and the rapid rate of new technology introduction into product development threaten the possibility of success in automating product development processes and ultimately wide spread adoption of mass customization. In addition, frequent changes in the market environment or engineering tools and processes, make it difficult to maintain valid automated processes after they have been created [3].

This work presents a potential solution to reduce the risk associated with the custom software development investment. The proposed method is structured around the deployment of a web service-agent framework. Such a framework allows product development systems to be automated using dynamic methods which can reroute the path of a process workflow as changes occur to the system.

The development of a dynamic framework involves implementing single purpose tasks as web services and controlling the workflow process execution in a multi-agent system. Using this method, decision and execution paths of the process framework can be determined dynamically as the objectives change. Multi-disciplinary optimization, sensitivity studies, uncertainty propagation, artifact generation, and design studies could each potentially produce different process systems based upon the paths required to accomplish the objective. The agent framework must be able to not only identify and construct these different processes, but provide a flexible execution method.

The general concept presented in this thesis is to develop a framework that enables “plug and play” capability for integrating automated modules related to specific tasks in an engineering process. This framework is developed once and used for all subsequent automation projects, thereby reducing the software development to identification of task modules within an engineering process and the implementation of these task modules in a “plug and play” form. Strategies and methods for developing the overall framework, identifying task modules within an engineering process, and preparing the tasks in a “plug and play” form are also presented.

The goal of this research is to establish a foundation for a dynamic, automated framework for product development through the following objectives:

1. Create a flexible design system built on web service and agent technologies.
    a. Define the system agents and identify the required process services.
    b. Develop a method for dynamic web service-agent integration.
2. Demonstrate feasibility of the proposed system applied to a generic application.
3. Demonstrate feasibility of the proposed system applied to a specific engineering application.

CHAPTER 2 LITERATURE REVIEW

2.1 The Product Design Generator

The Product Design Generator (PDG) developed by Roach et al is "a computer-based tool that automatically creates all of the design artifacts and supporting information that are necessary for the design of a product that is customized to meet the needs of a specific customer" [4]. Essentially, the PDG is a systematic approach to the creation of automated design modules for mass customization. This approach requires a complete description of the product development process, which along with business best practices and company knowledge can be used to transform unique customer requirements into a final product.

The PDG plays a central role in the development of the dynamic web service-agent system for engineering design that is outlined in this paper. For example, identifying the system services and agents requires following the process mapping strategies defined in the PDG methodology. In addition, many of the behaviors and services provided by system agents are simply an automated implementation of PDG construction practices.

2.1.1 Defining the Product Transformation Schematic

Developing a PDG requires that a Product Transformation Schematic (PTS) be created for a product design within a specified envelope of variation. The PTS is defined by a specific sets and process maps that are used to transform customer requirements into a range of products. The idea of a PTS is similar to the definition of a mathematical function, a single method extracted from an infinite number of solutions for transforming one set of numbers (the domain) into another set numbers (the range). Like a mathematical function a specific PTS can be realized by an infinite number of different PDGs. As a result, each PDG is a unique implementation for a specific product type [5].

At the highest level, the transformation of customer requirements into detailed designs can be represented by the function:

$$F(\Phi) = \Omega \tag{2.1}$$

where F is the product transformation function, Φ represents the specific product design requirements, and Ω is the required set of design outputs [4]. In order to provide the details necessary for PDG construction F, Φ, and Ω are deconstructed into more specific subsets and intermediate mappings. Figure 2.1 shows a schematic representation of these elements.

The domain Φ, the top level product design requirements, is deconstructed into two specific subsets: $\boldsymbol{C}$ and $\boldsymbol{K}$ [5]. The subset $\boldsymbol{C}$ is the customer specific requirements for the product design which typically includes performance metrics or limits. $\boldsymbol{K}$ is the subset which identifies company knowledge or conventions. This information is usually defined by company best practices or policies that are required to successfully design a product [5].

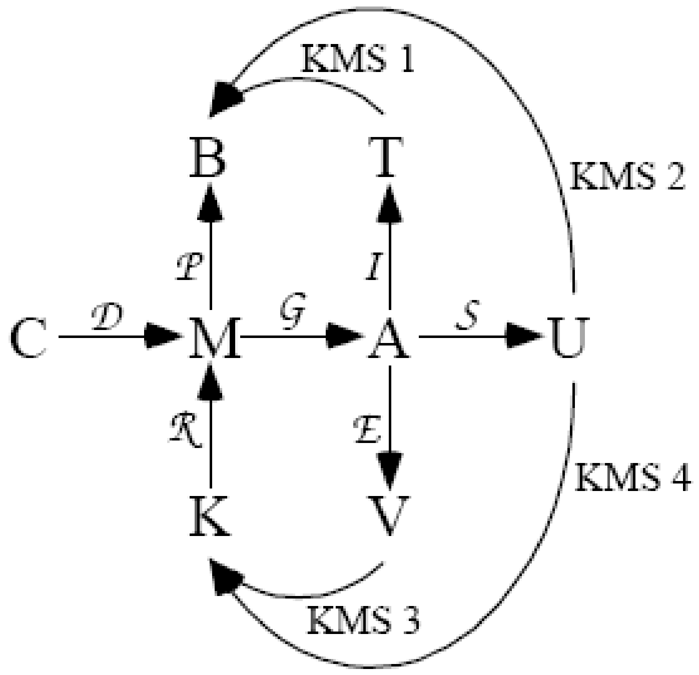


Figure 2.1 A schematic representation of the PTS

The range **Ω**, the set of required design outputs, is reduced to five subsets: ***B***, ***A***, ***U***, ***T***, ***V***, and ***M*** [5]. ***B*** is a subset of metrics that measure product behavior. Often some of the parameters contained in subset ***B*** are the same as others specified by the customer in subset ***C***. In this case, optimization and other iterative strategies are employed during process execution to reconcile any discrepancies between the parameter values. Subset ***A*** is all necessary product artifacts required for production, marketing, and support of a product. This subset might include manufacturing instructions, solid models, installation drawings, data sheets, etc. ***U*** is defined as the subset of all final customer deliverables, which could include technical product specifications, the package product, any documentation required for customer use, etc. The subset ***T*** is comprised of test and validation metrics for the product. ***V*** is the subset of all vaulted parameters, artifacts, or deliverables. The master parameter set, ***M***, is the primary repository for the controlling parameters for all possible configurations of a given product. It is important to note that many of these subsets have a coupled relationship similar to that with ***B*** and ***C***. The schematic in Figure 2.1 shows a general relationship between upstream and downstream subsets, where the downstream sets ***U***, ***T***, and ***V*** update the upstream sets ***B*** and ***K*** [5].

The primary relationship between the subsets of Φ and Ω is defined by the intermediate mappings that make up the transformation function *F*. The design map ***d*** transforms customer input into appropriate master parameters. Company rules and knowledge are also introduced into the master parameter set through another mapping ***r***. The map ***b*** is responsible for determining the predictive metrics from the master parameters. Generation of artifacts form the master parameters is a function of the design map ***g***. The testing and validation process is represented by the map ***i*** and the vaulting of information is handled by the map ***e***. The design map ***s*** represents the final transformation of artifacts into customer deliverables [5]. Identifying the members of the individual subsets and the processes represented by the intermediate maps is a critical step in the construction of a PDG.

2.1.2 Constructing the Product Design Generator

The PDG is constructed by (1) selecting a design concept and determining the layout for the subsets and intermediate mappings, (2) creating detailed process plans for the intermediate mappings, (3) rectification of the master parameter set, and (4) constructing reusable design modules and integrating them into a common framework.

The first step in constructing the PDG is selecting a design concept. From this concept and the desired design envelope the parameters of the subsets ***C***, ***B***, ***A***, ***U***, ***T***, and ***V*** can be determined. These subsets are generally enumerated first because they can be readily determined from engineering judgment, best practices, company or international standards, or past experience developing other similar products. Once these subsets have been constructed the intermediate mappings must be determined. This requires that intermediate mappings are further refined into individual parametric process maps [5].

Determining the parametric process maps leads to the next step – creating layouts for the detailed mapping plans. This is a process of outlining the different steps required for each map, defining necessary parameters, and identifying both internal and external relationships between parameters, product features, and other mappings. Constructing detailed process plans ensures that the intermediate mappings will be flexible and reusable. Once the subsets and intermediate mappings are determined the master parameter set can be compiled. This third step in constructing a PDG provides a repository of all parameters that define each unique design. When vaulted as part of the PDG process, the master parameter set along with the PDG implementation will provide sufficient information to recreate a customized design [5].

The final step in creating a PDG is the construction of reusable modules. Unlike the other construction steps, the actual PDG implementation is not strictly defined by Roach et al. Past PDG systems have been realized in a number of different frameworks including simple Visual Basic applications, ASP.Net web pages, and a distributed framework using web service technologies [5] [6]. Once the process mappings are implemented as reusable modules, storyboard models of the different lifecycle design phases are integrated with the reusable models to create the PDG [4].

The concept of a PDG is fundamental to the successful development of a web service-agent dynamic design system. The formal expression of the design process through the PDG methodology provides an established approach to mass customization that has been used as the primary foundation for this research. The PDG approach allows the existing tools and methods to be captured and defined in a reusable form that will improve the productivity of the design process in a consistent and repeatable manner [4].

2.2 Multi-agent Design Systems

During the last decade the concept of multi-agent systems has become an increasingly important research field in both artificial intelligence (AI) and computer science. At the core of much of this research has been an exploration of the science of agent systems through both theoretical and experimental results [7]. As multi-agent systems have become better understood by a wider community not limited to just AI and computer science, agents theories have been successfully applied to engineering design systems.

One of the primary motivations for developing engineering systems that utilize agent-based technologies is the lack of flexibility and adaptability in current process automation systems. Lander explains that "[d]esign, in particular, is characterized by a constant evolution of software tools and techniques and by the need to respond rapidly to changes in the market and industry" [3]. Such extensive change makes software maintenance in engineering design a particularly complex and difficult task. It is estimated that traditionally, software maintenance consumes 50% to 80% of an application's lifecycle cost [3]. In an attempt to avoid this significant cost, traditional approaches to automated product development have been reduced to a narrow product definition that is not as heavily influenced by such a volatile environment.

The need for "diverse, highly sophisticated, and rapidly changing skills and knowledge" as well as a more flexible approach to engineering design makes multi-agent systems "particularly appropriate for knowledge-based design" [3]. Agent-based systems require minimal software changes to existing tools by "wrapping" legacy code with agent functionality, make process changes without altering system code, vault knowledge and data autonomously, present an open and well defined knowledge representation and behavior model, and are remotely accessible [9]. These flexible characteristics dramatically reduce the custom software development and future maintenance associated with traditional automated engineering systems.

Because multi-agent systems have many far reaching benefits, it is not surprising that a considerable amount of research has been focused on developing agent-based applications for engineering design. Many of these systems, unfortunately, are no longer in study or development. Wooldridge et al explain that misunderstanding the pragmatics of multi-agent system development results in developers “needlessly repeating the same mistakes, with the result that, at best, resources are wasted – at worst, projects fail” [7]. Despite the lack of longevity in many agent-based engineering systems, it is important to review the different implementation attempts to improve future development projects. Shen et al provide a comprehensive overview of existing agent-based design systems, including historical context, important technical features and implementation details, significant results, and how the project is positioned with respect to other methods [10].

One of the first agent-based systems for engineering design was the Palo Alto Collaborative Testbed (PACT). PACT utilizes agent technologies to encapsulate existing engineering tools in order to provide a communication interface using the agent languages KQML (Knowledge Query and Manipulation Language) and KIF (Knowledge Interchange Format). Agents within this system worked with facilitators as “middle-men” to coordinate the configuration of a simple robotics manipulator from a design catalog. The end result of this project was not so much an actual environment for agent-based design, but a mechanism for distributing reason to engineering design tools [10]. PACT spawned the development of a number of other design systems that focused heavily on studying the benefits of different agent frameworks for collaborative engineering between groups of human agents.

The Distributed Intelligent Design Environment (DIDE) represents another significant study in agent-based engineering design systems. This system uses a peer-to-peer approach to a multi-agent system where agents can be either human specialists or automated computer modules. These agents are integrated into a business-like framework where some agents act as project managers and others as lower-level managers overseeing smaller groups of agents.

DIDE is defined by a few core characteristics. First, the system does not run in an autonomous fashion. The project manager is the agent responsible for initiating action in local groups. These tasks must then be delegated by local group managers to individual agents some of who will operate automatically. Second, agents in this system are purely reactive. There is no form of collaborative planning to achieve global system goals. Third, local groups are not responsible for maintaining global consistency. The project manager is responsible for reconciling discrepancies. Fourth, agents maintain their own knowledge base and have the ability to communicate with any other agent in the system. These agents are capable of adding themselves to the system by broadcasting their abilities and removing themselves from the system by broadcasting a removal message. Individual agents use the information from other agents to update their own knowledge base [10].

DIDE demonstrated several key advantages to multi-agent design systems: (1) multi-agent systems are an appropriate solution for creating dynamic and flexible design systems; (2) autonomous and independent agents allow for easy integration with legacy tools; (3) human agents can interact successfully with computer agents; (4) agent-based systems provide efficiency through parallel design. Along with the benefits, however, were some notable disadvantages: (1) testing the overall behavior of a peer-to-peer agent system is prohibitively complex; (2) the complexity of the development process prohibits applying this strategy to small scale projects; (3) interactive tools are difficult to integrate into the system; (4) the system introduced problems with consistency of design representations, reliability of communication, and sustainability of cooperation processes into the process [10].

Other notable agent-based design systems include ACDS (Automated Configuration-Design Service), a distributed, automated synthesis approach that uses an agent network to generate a space of all possible complete designs and then narrow the space to feasible designs only [11]; A-Design, a multi-agent system design tool that utilizes multi-objective optimization and automated design synthesis for the conceptual design of electro-mechanical devices [12] [13]; CADOM (Component Agent-based Design-Oriented Model), uses a hierarchical design-oriented model to build agents that carry out the functions of a designer [14]; and RAPPID (Responsible Agents for Product-Process Integrated Design), a agent-based design system that uses market dynamics among distributed design team members to coordinate set-based design of a discrete manufactured product [15].

CHAPTER 3 BACKGROUND

3.1 Service-Oriented Architecture (SOA)

A service-oriented architecture (SOA) is one of the key technologies which enable the development of a dynamic framework for product development. In recent years much of the progress of SOA concepts and technologies has been driven by a desire to reduce network-based application development time while also increasing the flexibility and connectivity of applications. As a result, much of the information technology industry has embraced an SOA approach, specifically one focused on web service standards, as the solution to increased productivity in enterprise computing. It is expected that adopting an SOA approach will open the door for flexible, secure, and reliable communication over both internal and external networks.

Service-oriented architecture is a development approach to connecting applications, often exposed as services made available over a network, so they can communicate and share functions in a widespread and flexible way [16]. A service is defined as a specific task or functionality implemented in such a way as to facilitate consumption by clients in different business processes or applications. Services can also be defined by varying levels of granularity. Fine-grained services may represent a single, well-defined task such as a mathematical operation or database query. Similarly, course-grained services can be used to represent the functionality of a software application or an entire business process. Course-grained services can also be used to represent a directory of services and any necessary details required for a client to consume the service.

One of the essential characteristics of SOA is the idea of loose coupling between services and clients. Loose coupling requires that the service have a well defined interface (the “what”) which is separate from the implementation (the “how”) [6]. This approach creates a flexible environment where the client is not required to understand implementation details such as the platform the service runs on, the language it is programmed in, or what additional processing might be required for the service to return a result. The client must only understand how to interact with the service interface. Tightly coupled services often share semantics, libraries, or state resulting in a system that is difficult to maintain as environments or needs change [16]. Document-oriented services further enhance the concept of loose coupling by accepting a document as an input, rather than a more specific, often restrictive, data type. In this case the service is responsible for interpreting the context of the document and executing appropriately.

In addition to the flexibility that results from implementing an SOA, applications are also able to more easily scale as demand increases or decreases. Because loose coupling of services results in fewer dependencies between clients and services, asynchronous communication is possible. Therefore, services are able to scale to meet required loads without introducing the increased lag or delay experienced by a tightly coupled system dependent on synchronous communication.

The reusability of services is one of the important benefits driving widespread adoption of SOA concepts. Because services separate the interface from the implementation, the actual code exposed as a service can be reused as interface requirements change. Functionality that is loosely coupled in this way is more likely to be reused in future applications than tightly coupled functionality built into a specific application. Also, because the interface is the only part of a service exposed to client consumers, legacy applications as well as those developed by business partners can be used and reused more readily.

Essential to the success of service reuse is interoperability between services and clients. Interoperable services are able to communicate with both clients and other services regardless of the specific service platform, system, or programming language. The WS-I basic profile for web services defines the core technologies and concepts that ensure web services are interoperable with each other. Strong backing from the information technology industry for the WS-I basic profile helps guarantee the interoperability of web services developed to this standard [16].

The flexibility, scalability, and reusability of services created in a SOA provide a cost effective solution to integrating business applications. An SOA approach allows for reuse of legacy applications, which could potentially include applications that were previously unusable, by adding a standard service interface for client interaction. In addition, this SOA creates a solution for integrating business partner applications with minimal custom development. More important to this research, however, a SOA contributes to reducing the risk associated with developing a product development system that is dependent on custom software to integrate various CAx tools. Product development systems based on SOA and web service standards require less initial analysis and unique code in developing applications custom engineering applications. Loosely coupled engineering services also provide flexibility and scalability necessary for extended application life and reduced maintenance costs.

3.2 Web Service Standards

Web services provide an effective, standards-based approach to SOA. The core web service standards defined in the WS-I basic profile [17] include: Simple Object Access Protocol (SOAP), Web Service Description Language (WSDL), and Universal Description, Discovery, and Integration (UDDI). Each of these standards is based on eXtensible Markup Language (XML) for describing and exchanging data over a network.

3.2.1 eXtensible Markup Language (XML)

XML [18] is a general purpose markup language which uses “tags”, similar to those used by HTML, to identify information in a document and organize it in a structured way. Unlike HTML, XML does not have a limited, inflexible set of tags, elements, or attributes available for use. As a result, an associated schema must be used to give meaning to the different XML components and provide a required structure for the data contained in the document. The schema creates a set of grammar or language rules that can then be used by either humans or machines to interpret and understand the XML document. Because XML documents must be well-formed and conform to the associated schema, they provide an effective data language for web services [16].

3.2.2 Simple Object Access Protocol (SOAP)

SOAP is an XML-based protocol for "exchanging structured and typed information between peers in a decentralized, distributed environment" [19]. While XML provides the meaning and structure necessary for exchanging data, SOAP provides a protocol for communicating the message "payload" and any additional instructions or supplemental content [16].

SOAP defines a message as a one-way transmission between SOAP nodes. This message consists of a SOAP envelope which contains an optional SOAP header and mandatory SOAP body. The header is an extension to the communication protocol which assists in routing the message through intermediate nodes for additional processing to its final destination. The body contains the end-to-end information intended for the final recipient. SOAP standards are not only platform independent, but they are not tied to a specific communication protocol. As a result, both SMTP and HTTP, which work well in today's existing internet structure, are frequently used with SOAP.

3.2.3 Web Service Description Language (WSDL)

WSDL is another key standard defined by the WS-I basic profile. WSDL is an XML approach to defining how to communicate with a given web service. A WSDL document is essential for a client to understand how to make a service request. Specifically, the WSDL provides protocol bindings and message formats for a web service interface through an XML schema. By accessing the WSDL document in web service registry, a client can use XML standards, including SOAP, to determine where a web service resides, its function, and the process for invoking it.

A WSDL document defines a web service as collections of abstract endpoints or ports. Other abstract elements are used to represent service data and actions. Messages are abstract descriptions of the data being exchanged and a port type is an abstract collection of operations supported by one or more endpoints. Once these items are described abstractly, they are bound to a concrete network protocol and the data format specifications for a particular port type. The WSDL document allows a client to access the binding and network address for each port, and call a web service operation according to the specified protocol and message format [16].

3.2.4 Universal Description Discovery and Integration (UDDI)

The UDDI specifications define "how to publish and discover information about services in a UDDI-conforming registry" [16]. The UDDI specifications are comprised of a definition for a UDDI schema and a UDDI API. The UDDI schema uses XML data structures to describe web service registry entries. The API provided through the UDDI specifications defines the appropriate structure and content for SOAP messages that publish in the registry or search for available services. Like UDDI, ebXML is another web service standard that includes a registry.

Many of the current registry technologies including UDDI and ebXML standards do not provide a method for autonomous discovery and integration with web services. This is primarily the result of a lack of machine understandable information in the registry entry descriptions. Potential solutions to this issue are discussed later in the context of semantic web services.

3.3 Multi-agent Systems

3.3.1 Agent Definition

Like web services, agent technologies are another important enabler of dynamic product development. Agent-based systems are able to solve problems that are too large for a single-resource limited system; facilitate the interconnecting and interoperation of multiple existing legacy systems; provide solutions where the expertise and information is distributed; and enhance system speed, reliability, extensibility, and the ability to tolerate uncertain data and knowledge. Many engineering systems rely on a vast catalog of legacy software, extensive product and/or knowledge databases, and parametric CAx tools for development and manufacturing. Agent technologies provide many of the tools necessary for linking these elements together in a dynamic, flexible system.

An agent is defined as “a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives” [20]. While this characterization is generic enough to apply to most agent implementations, there is no universally accepted definition for the term “agent”. In fact, “agent” has become a buzzword that is often applied erroneously to expert systems, artificial intelligence, object-oriented programming, and web service concepts [7] [20] [21]. Despite these discrepancies, it is generally accepted that agent autonomy is key to understanding this type of agency [22].

Wooldridge makes several important points about agent definition [22]. First, it is important to distinguish between "agents" and "intelligent agents". The term agent represents a more generic definition that can be extended by different properties or behaviors. An intelligent agent is one such extension defined by flexibility, meaning that it is:

- <u>Autonomous</u> – Agents control both their individual state and behavior.
- <u>Reactive</u> – Agents are able to perceive changes in their environment and respond in a timely manner.
- <u>Proactive</u> – Agents demonstrate goal-oriented behavior by taking initiative to meet objectives.
- <u>Social</u> – Agents are able to communicate and work with other agents to satisfy design objectives.

It is also important to understand that intelligent agents are not limited to computer systems. Any entity that can meet these conditions, including humans, can be treated as an intelligent agent [22]. For this research, the basic definition of an intelligent agent will be used in developing an agent-based framework including extending agent concepts beyond software applications.

Wooldridge also notes that the general definition of an agent is not tied to any specific agent environment [22]. Russell et al [23] characterize an agent environment as:

- Accessible or Inaccessible – An accessible environment enables an agent to accurately determine the environment's state at any given time.
- Deterministic or Non-deterministic – A deterministic environment is one in which there is a specific, guaranteed outcome for an action.
- Episodic or Non-episodic – Agents that react in an episodic environment must determine if its actions will have an impact beyond the current episode.
- Static or Dynamic – Dynamic systems require the agents to adapt to change.
- Discrete or Continuous – A discrete system has only a limited set of potential agent actions.

Because agents can exist in a wide variety of environments characterized by properties similar to those listed above, the complexity of implementing an agent-based systems is necessarily coupled with the agent environment.

On the most basic level an agent interacts with its environment by processing its perceptions through sensors and acting on the environment through effectors [20]. This relationship, as shown in Figure 3.1, is similar to the operation of a home thermostat which acts as a very simple agent. The thermostat uses a sensor to determine the temperature in a room. The thermostat then manipulates the temperature controls to maintain or change the room temperature to match the desired temperature specified by the homeowner. Most systems a developer will encounter require a more complex agent definition and exhibit far less control over their environment than is demonstrated by the thermostat example. However, this example does show the core functionality of both sophisticated and unsophisticated agent-based systems. Agents that utilize this same core functionality have been used to create autonomous, flexible systems for general distributed computing, network management, goal-oriented web search, e-commerce, manufacturing process control, air traffic control, UAV control and cooperation, distributed behavioral models, population simulations, and many other varied applications.

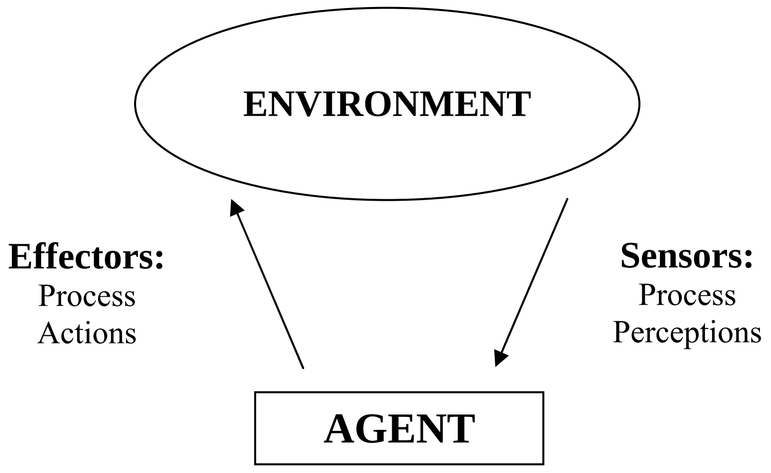


Figure 3.1 Agent interaction with the environment

Providing for agent communication is essential to developing more complex multi-agent systems. Several standard languages have been proposed for use in multi-agent systems. One of the more common languages is the Knowledge Query and Manipulation Language (KQML) that was developed as part of the ARPA Knowledge Sharing Effort [10]. This messaging protocol allows agents to communicate information by annotating messages to describe specific requests. Another language in wide spread use in the Agent Communication Language (ACL) [24]. ACL is standard maintained by the Foundation for Intelligent Physical Agents (FIPA) that functions similarly to KQML. The work present in this paper does not attempt to employ a standard language for use in the case study examples.

3.3.2 Web Services and Agents

The W3C Web Services Architecture specification defines agents as, "…the running programs that drive web services – both to implement them and to access them as computational resources that act on behalf of a person or organization" [25]. This definition of an agent identifies one of the primary motivations for implementing multi-agent systems. According to Wooldridge, agents are primarily "responsible for mediating between users' goals, and the available strategies and plans" [21]. Agents accomplish this by creating composite web service workflows and consuming individual services that satisfy design objectives and goals.

Although web services and agents both provide a means for encapsulating business or application knowledge, they differ in that agents "do not simply expose functionality as methods over a fixed protocol" [26]. Rather, agents "offer multiple services, or behaviors, that can be processed concurrently and activated specifying goals" [26]. The abstract, goal-driven behavior is unique to the definition of an agent. Unlike web services which provide functionality through simple executable methods, agents that act intelligently use knowledge to react to and act on their environment autonomously and proactively.

Web service-agent systems have recently become the focus of a significant number of research activities. One of the more apparent issues with web service implementation is the creation of a framework that provides autonomous selection and consumption of the services. As agent technology has matured, many have begun to address this problem and investigate the use of an intelligent agent-based system as a potential solution.

JADE (Java Agent DEvelopment Framework) is one of the most pervasive agent development platforms in use today [27]. JADE is an open source, middle-ware solution for developing peer-to-peer agent applications that comply with FIPA (the Foundation for Intelligent Physical Agents) specifications for agents and multi-agent systems [28]. Recently this platform has been extended by Greenwood and Calisti to provide development tools for web service-agent interactions through the Web Service Integration Gateway Service (WSIGS) [26].

The primary goal of the WSIGS extension to JADE is to provide a "transparent, bidirectional access form/to web services to/from agent-based services" [26]. Agents implemented in this system are able to:

- Redirect the system to other web services when the originally requested service becomes temporarily unavailable.
- Aggregate several web services to function as an individual service interface.
- Integrate web services into preexisting agent systems.
- Manage and perform administrative tasks for large groups of web services.

Web services and agents developed with this extension use existing standards and technologies. The primary function of WSIGS is to provide a common interface for translating and forwarding web service-agent communication.

Dickinson and Wooldridge have expressed concern over the confusion that ultimately results to the core definitions of web services and agents with WSIGS style bi-directional integration [21]. On one hand, in order for web services to invoke agent behaviors it is implied that the agents must expose a fixed, deterministic behavior. This approach “violates the presumption of the autonomy of the agent” and brings into question the validity of referring to this component as an agent [21]. Likewise, if the agent behavior is not fixed a web service must adopt ability to respond to the agent’s autonomous responses. Conceptually this is closer to the definition of an agent rather than a web service [21].

Along with their critique of WSIGS, Dickinson and Wooldridge identify some important behaviors of web services and agents in an integrated system. First, agents and web services share motivation to create flexible and adaptable systems, but are nevertheless distinct in their implementation and functionality. Second, agents are the responsible party for composing complex service workflows from the individual, atomic web services. Third, autonomy is only represented at the agent level. Finally, agents are capable of planning by decomposing high level goals in specific sub goals [21].

Other important research focused on web service-agent interaction includes the concept of extending UDDI with the DARPA Agent Markup Language (DAML) presented by Maximilien and Singh [29]; an approach to adaptive workflow that uses the Business Process Execution Language for Web Services (BPEL4WS) to define the initial structure of a multi-agent system [30]; a system that uses a workflow agent to dynamically compose web service workflows by using semantic descriptions of the services to find and match service inputs and outputs [31]; and the use of the Web Ontology Language for Semantic Web Services (OWL-S) to create middle agents to assist in dynamically discovering and connecting with appropriate web services [32].

3.3.3 Design System Infrastructure

Although agent technologies are inherently flexible, the choice of an infrastructure technology "affects the degree of asynchronous, concurrent, or persistent activity that can occur; the way information is stored and shared; and how agents communicate" [3]. As a result, the infrastructure used to build a multi-agent system must be carefully selected to provide the desired behavior from the system agents. Three basic system infrastructures will be discussed: an agent network, a federated environment, and agent-based blackboards.

An agent network is essentially a peer-to-peer network for a multi-agent system. The agents in the network are required to have a global knowledge of the complete system for communication and operation. They must know where and when messages should be sent, what other agents are available in the system, and what services that are provided by these agents [3]. The DIDE system mentioned earlier in this paper is one example of a peer-to-peer approach for developing agent networks. Like the DIDE application, there is system-wide duplication of knowledge that must be embedded in all system agents. This duplication makes this type of agent network inefficient for large systems. Figure 3.2 shows an agent network comprised of two local networks.

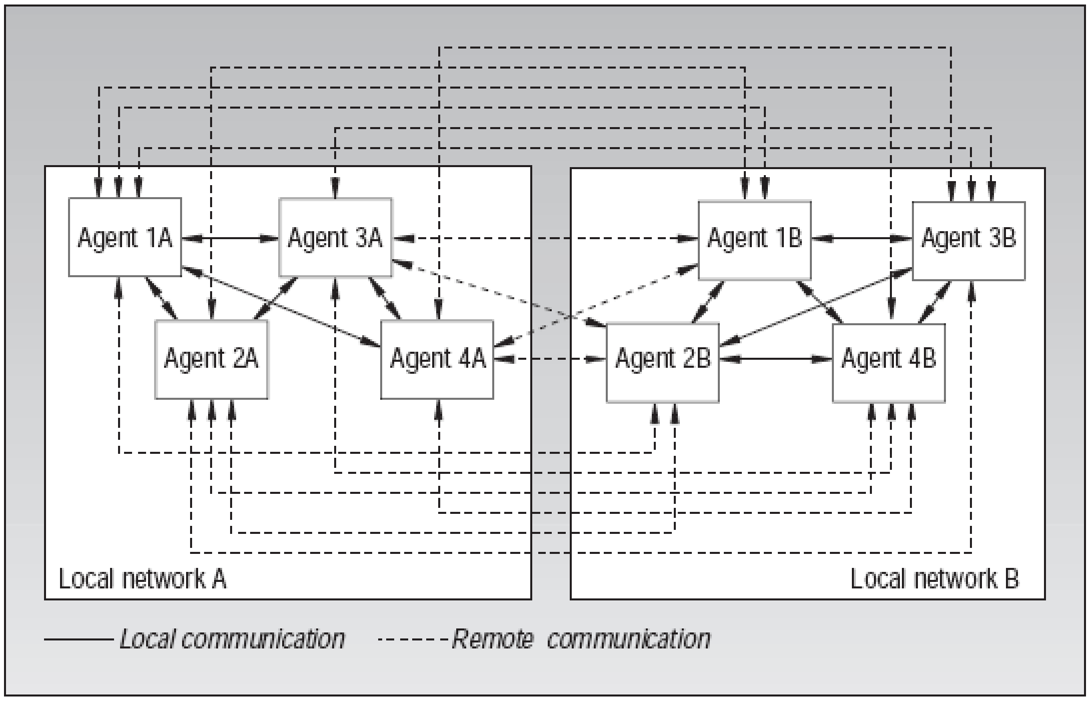


Figure 3.2 A schematic representation of an agent network [3]

The second infrastructure that can be used for multi-agent systems is a federated network. The federation operates through agent facilitators who handle all network connectivity and message routing. Facilitators also serve as a local repository for knowledge about individual groups of system services. A service makes a request through facilitators which in turn respond by searching for appropriate agents to satisfy the service request. A service provider can also work through the facilitator to find additional services that require their functionality [3]. Because federated systems do not require global knowledge to be shared with all agents these system types will be more flexible. Figure 3.3 shows an example of an agent federation.

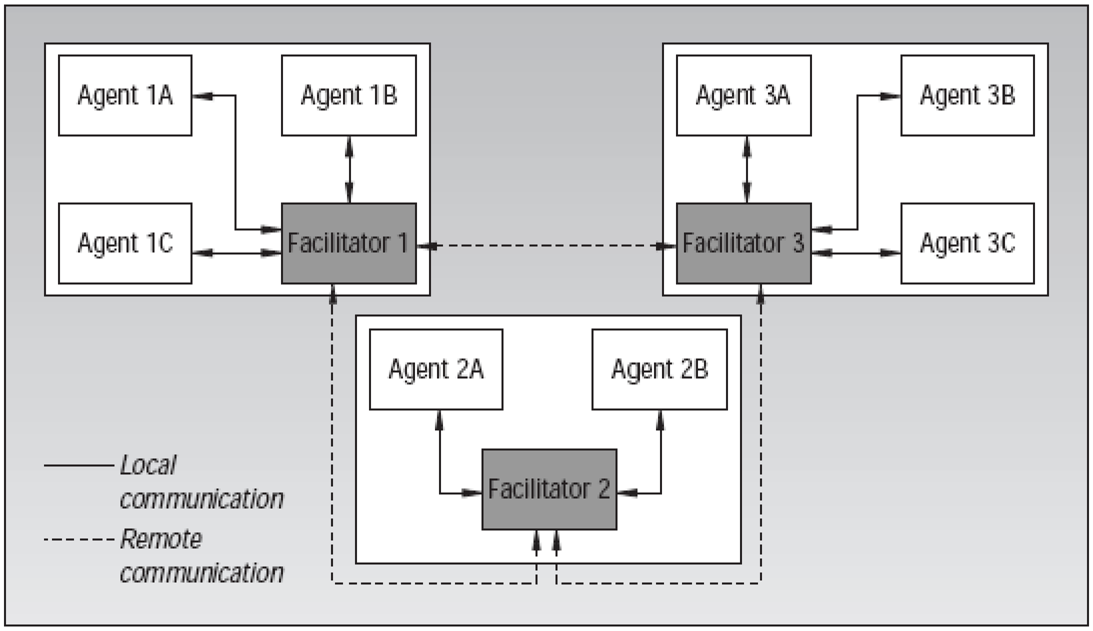


Figure 3.3 A schematic representation of an agent federation [3]

The final infrastructure type is an agent-based blackboard. Blackboards are very similar to federated infrastructures in that create localized groups of agents. While all federations share in the same database, the blackboard approach provides a data repository for design data and control knowledge within each local group. A network controller and control shell are also provided for local groups to communicate remotely. Figure 3.4 shows a blackboard infrastructure.

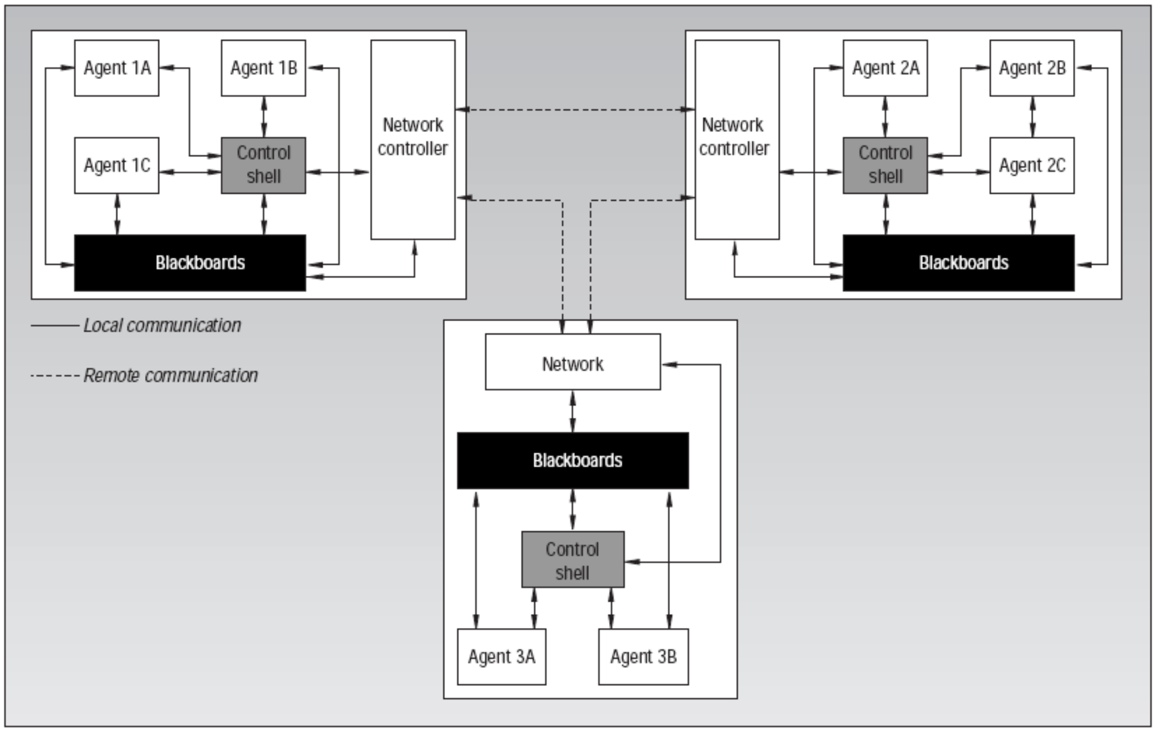


Figure 3.4 A schematic representation of an agent-based blackboard [3]

3.4 Semantic Web & Ontology

As web-based technologies such as web services and autonomous agents have become more mature and widely accepted, the Internet has moved beyond a simple tool for communicating textual and graphical information, to a provider of services enabling automation and interoperation. Where the Internet was once focused primarily on delivering content for human interpretation, recent development trends have focused on creating "web-enabled" applications and physical devices that capitalize on Internet-provided services. One purpose of a Semantic Web – an extension of the current Internet structure which attempts to communicate meaning (semantics) in a machine understandable form – is to enable reliable, large-scale interoperation between web services, autonomous agents, and "web-enabled" applications and devices by making service information computer interpretable [33].

Fundamental to understanding the Semantic Web and ultimately communication between services and agents is the concept of ontologies and other data models that can be used to represent some domain such as controlled vocabularies and hierarchical taxonomies. Ontology is frequently defined in relation to the Semantic Web as "a set of knowledge terms, including the vocabulary, the semantic interconnections, and some simple rules of inference and logic for some particular topic" [34]. Regardless of the formal language of expression, ontologies generally describe information with:

- Classes – an abstract group, set, or collection of objects
- Individuals – an instance of a particular abstract object or concept
- Attributes – object defining properties, features, characteristics, or parameters
- Relations – parameters defining association between objects or concepts

Because ontology is often used to represent specialized concepts that may have specific meaning to a particular sub-domain, high-level domains are necessary to define essential concepts and merge the vocabulary of different sub-domains into a more generic representation [10].

Web service and autonomous agent systems benefit from using ontologies to "decipher the content of exchanged messages" [10]. One of the problems with current web service implementations is the difficultly in dynamically discovering and consuming the service without human assistance. The public UDDI registry that was formally closed in January 2006 by IBM, Microsoft, and SAP [35] attempted to provide a more universal approach to automatic web service integration by using the UDDI standards to catalog business services. Developers could access the registry and search for services that would meet a desired objective. However, without a formal method for providing semantic information, human interpretation of the web service descriptions was still required.

Unlike previous approaches, an ontology-driven, semantic markup of registered web services would enable a machine to automatically:

- Select web services for consumption based on a set of user-driven requirements.
- Understand and independently act on input/output requirements and execution details of a particular service.
- Interface with multiple web services to provide results for more abstract objectives.

The Web Ontology language for Services (OWL-S) [36] is a markup language intended to facilitate these results. Like its predecessors (e.g. DAML, OIL), related technologies (e.g. XML, RDF), and other unique approaches (e.g. Semantic Annotations for WSDL) [37], OWL-S is an attempt to represent meaning and semantics with machine-interpretable content.

OWL-S describes web services with a service profile, service model, and service grounding. The service profile identifies what the service does. In addition to the actual function of the web service, the service profile describes any limitations or special requirements for a specific web service. Not only does this allow agents and other services to access service profiles from a web service registry and use this information to identify a service that will satisfy a request, but it provides sufficient information for this to occur autonomously. The service model details how to use a service and what results can be expected. It also provides "the semantic content of requests, the conditions under which particular outcomes will occur, and, where necessary, the step by step processes leading to those outcomes" [36]. The final piece of structured information provided by OWL-S web services is the service grounding. The grounding is the specific information necessary to access the web service, including the required communication protocol, message formats, and the specific input and output data types expected by the service.

While formal markup technologies are still limited in use and maturity, other less sophisticated approaches are beginning to emerge on the Internet. Web sites such as del.icio.us (a social bookmarking service), Flickr (an internet photo sharing service), YouTube (an internet video sharing service), GMail (an internet email client), Technocrati (a weblog search service), and many others all use metadata "tags" to label and describe information and content (i.e. photos, videos, music, etc.). The process of "tagging" does not adhere to a formal ontology or taxonomy, but rather uses a more eclectic, collaboratively generated method of modeling data referred to as folksonomy. Although categorizing information in this way can create conflicts and confusion, it does provide an important step in migrating to the more structured idea of the Semantic Web.

Creating ontologies for a web services or agent-based systems that are focused on a specific function or utility requires a different approach from the more general web situations since the web is much larger in scope, often requires less complex information, and does not have the same requirements for automation and adaptation. In principle, a universal ontology could be used for knowledge sharing, however, differing system requirements, such as those already outlined, show that a universal approach is not a practical approach [10]. For many systems, particularly the dynamic product development framework presented in this paper, a simplified, less-flexible approach to ontology is sufficient.

CHAPTER 4 METHOD

4.1 Agent-based Control Framework

The first step in creating a dynamic product development system is to define an agent-based control framework. The design of individual agents is highly dependent on how a specific agent control framework is implemented. Ultimately a framework will have a significant impact on the location of the system knowledge base, the method for both local and remote communication, and the complexity of design problems that can be addressed. For example, a peer-to-peer agent network will require a sophisticated agent with knowledge of the entire system and ability to communicate with any other available agents. While this method may have many different practical applications, the framework can end up duplicating knowledge and ability in different agents making some large processes too inefficient for the peer-to-peer system.

For this research a more simple approach to an agent-based framework has been adopted. Avoiding more complex development of multi-agent systems enabled greater focus on the practical engineering application of the general concepts rather than just the nuances of developing an agent-based system. As a result, generic agent framework was created that is not coupled to any specific programming tools or engineering processes, and system agents are limited in knowledge and communication ability. These agents are limited to providing only the services needed to generate an automated engineering tool enabled by web service-agent interaction.

The control agents developed for the dynamic product development framework serve the system in three ways:

- Managing a web service registry and identifying potential services
- Configuring process maps of all potential workflow paths
- Executing an automatically selected workflow path to meet the system design objectives

Implementation of these tasks was accomplished by creating three types of control agents: a web service registry management agent, a workflow configuration agent, and one or more workflow execution agents. The detailed function of each of these agents is discussed in this section.

4.1.1 Web Service Registry Management Agent

The web service registry management agent (MA) is responsible for monitoring and updating system web service registries. The actual implementation of the MA will differ depending on the type of registry or web service technologies used by the system. For example, existing web service registries such as UDDI and ebXML registries often do not contain enough information for successful autonomic service selection. In this case the MA must be capable of providing the system with additional semantic information to will facilitate automatic selection and execution by other system agents or services.

One of the more simple methods for making semantic information available would be for the MA to maintain and update a supplemental database which would provide the enhanced web service. The MA would be responsible for gleaning web services semantics through description languages such as OWL-S [36] and/or by calling web service methods specifically designed to provide detailed service information. The system framework developed for this research uses the latter approach to gathering web service details for the web service registry.

In addition to registry management, the MA also has the responsibility to notify the workflow configuration agent of any new services, removed services, or changes to existing services that could alter an established workflow (e.g. changing the output type of a critical service, moving a service to another network location). This function requires that the MA have some knowledge of how the framework establishes a specific process workflow in order to identify impacting changes.

4.1.2 Workflow Configuration Agent

It is the responsibility of the workflow configuration agent (CA) to determine the potential workflow paths for the services registered with the system. Engineering workflow is defined for this research as the process of executing individual engineering tasks in a specific process order. Specifically, this workflow is established independent of the agent framework by the mappings of the PTS [4]. The CA identifies the workflow paths by determining the associated input and output sets of the registered web services and then backwards mapping to link the services in a workflow pattern similar to the approach presented by Laukkanen et al [31]. By following this approach, all possible workflow paths for a given product development process can be determined. The CA maintains a system workflow repository by adding new process maps and removing workflow paths with invalid services after receiving updates from the MA.

In addition to determining a specific workflow path the CA may also be required to determine workflow cost (i.e. resources, time, etc.) and overall process capability (i.e. process variation, input sensitivity, etc.). The CA uses additional semantic data to enhance more basic workflow information which can be stored along with the process map in the workflow repository. As new or altered web services create additional available workflows, the agent framework can use this semantic data to automatically select and execute the best process for a specific application.

Key to the success of the CA is the development of a system language for describing process workflow. Without a robust process language the system would not be able to understand workflows with complex bifurcation or nested sub-processes. The Business Process Execution Language for Web Services (BPEL) developed by Microsoft and IBM provides a standard format for defining the structure of a process [38]. BPEL is an XML based language that describes complex process structures, attributes, and external processes relationships. Process languages such as BPEL may be helpful for describing complex processes [31]; particularly processes that result require any recursive behavior or systems that employ sophisticated ontology.

While the actual process description language used by the CA is not important, it is critical that the CA does at least provide the minimum information required by the execution agents for dynamic web service binding and invocation. A simple process language was developed for this research that described a process by web services linked by similar inputs and outputs. Using a simple language of this type limited initial prototypes to linear processes with simple bifurcation.

4.1.3 Workflow Execution Agent

The workflow execution agent (EA) is responsible for controlling the execution of a specific engineering process. While both the MA and CA have a fixed objective, updating the web service and workflow registries, the system framework may have any number of different EAs each with a different execution objective. These objectives could include multi-disciplinary optimization, sensitivity studies, uncertainty propagation, or artifact generation. Because agent technologies enable intelligent and autonomous action, product development systems can capitalize on unused resources (e.g. network or computing system downtime) to train neural nets, explore design space, or improve individual agent knowledge base.

The communication and interaction between multiple EAs and other system agents must be handled by a multi-agent environment capable of managing agent interaction to meet any system-level goals specified for the product development process. EAs embody the intelligence needed for truly autonomous actions, while the MA and CA function to provide the semantic information required by the EAs. The EAs are the agents that answer the system-level engineering process questions (e.g. Which design provides maximum life? How do I minimize cost?) These top level questions are posed to the EAs which in turn explore the variety of workflow paths identified by the CA to find answers to these questions. This method of problem decomposition provides significant flexibility in posing the top level questions by answering the question with different workflow paths, technologies, or combinations of process task modules.

Improvements in process technologies or changes in process steps can be automatically implemented by the EAs once the CA is notified of changes to the set of available services by the MA, updates the workflow repository with new process maps, and then provides notification of change to the EAs. To illustrate this concept consider an engineering process for predicting deflection in some product component. The original process may include the use of traditional closed-form linear equations captured in a spreadsheet, and the initial workflow paths identified by the CA might include this spreadsheet implemented as a stand alone web service. If later in the product life it is determined that more sophisticated deflection results are needed, a finite-element technique might be developed as a web service and the module introduced into the system. A secondary workflow path would be identified by the CA which would include the finite-element module instead of the spreadsheet. Determination of which workflow to use would be determined by the EAs based upon criteria most likely linked to an engineering process question such as quality of prediction based on field data or empirical correlations.

4.2 Identifying and Creating Web Services

Determining the web services (i.e. process task modules) to use in the system framework requires following a theoretic process decomposition to extract the specific tasks associated with a given product development process. This decomposition involves identifying the associated input and output variables of a process task, backwards mapping the dependencies, and defining the tasks that convert the inputs to the outputs. Backwards mapping is a simple dependency resolution process starting with the desired result of the process and working backwards to identify the required inputs.

This decomposition links the inputs and outputs as well as the tasks into a single theoretical integrated workflow as described by Roach et al [4] in the formulation of the PTS. This procedure identifies the necessary low level process tasks, referred to by Roach as intermediate mapping functions, which link the system inputs and outputs. Each of the identified intermediate maps then must be structured and implemented as a web service. In this way, all necessary tasks for the specific product development process are defined, allowing complete coverage for the overall product development process. The integration of these web services into the specific process is left to the CA.

Once the services are identified, they need to be exposed for consumption by the agent-based framework. This can be accomplished by creating a web service for each process task module in the product development process. Although web service technologies are focused on set description and communication standards, there are a number of different methods for implementing these standards. The solution outlined in this paper does not require that any one particular solution be used. However, the selected web service implementation must provide for a dynamic method for binding to and consuming the web services. Some existing web service APIs and autonomous agent development tools such as the JADE platform provide the necessary resources for creating dynamic web service-agent interactions [26].

The CA must follow some type of system language in order to successfully map the potential workflow paths. A system language is a common syntax for defining the process task inputs and outputs. Decomposition and web service creation has a significant impact on the common system language that will be used to define the service inputs and outputs. In more complex systems a simple system language may not be sufficient for describing all potential links between services or may be overly restrictive. In this case, detailed system ontology can be used to create object-oriented structure for the system language. Once the system language is determined all web services must represent their different operations in this format, whether by exposing the information through a semantic description format or by reporting this information through executable methods.

CHAPTER 5 RESULTS

5.1 Case Study #1 – Ring Example

To better illustrate the proposed method, an example case study was created. In this case study, a generic engineering process was selected and modeled as a sequential process that connects four components together into a ring structure. This could represent an assembly of four parts, an interlinked analysis process, etc. In this example, the ring is constructed by assembling parts A-B-C-D in the correct order. It is important to note that part A cannot be added to C without first adding either B or D. A similar rule also holds for parts B, C, and D. Each action box shown in Figure 5.1 represents potential process tasks, services, or the action of particular agents. For the proposed method to be successful, the prototype system must be capable of finding all possible workflow paths, identifying and executing a workflow path based on some criteria, and be able to handle creation, removal, or changes to the system services without requiring any additional system-level software modifications.

The schematic shown in Figure 5.1 represents the results of the backwards mapping process and decomposition which can now be used to identify the tasks or services to be created. The process of creating the ring structure can be decomposed into four separate process tasks: Add A, Add B, Add C, and Add D. Each of the process boxes in the schematic can be subdivided into these sub-functions. In most cases, only the "leaf-node" functions need to be captured as services.

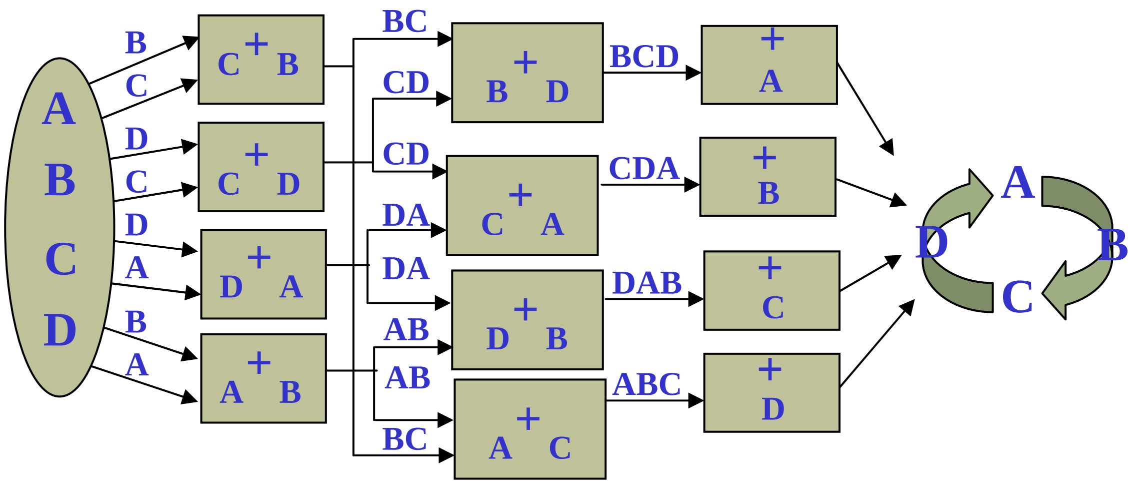


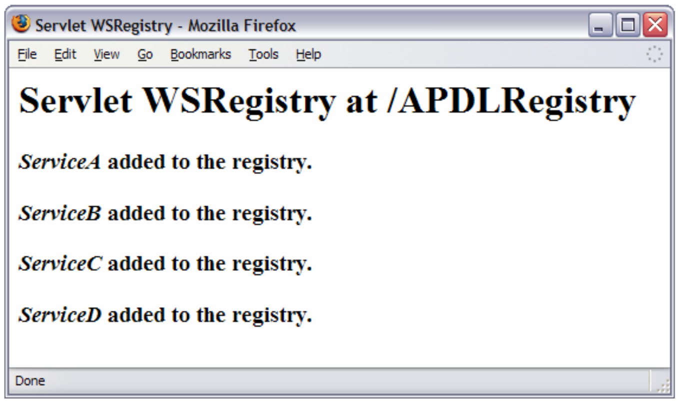
Figure 5.1 Process schematic for creation of basic ring structure

For this study, the service action was represented by a simple mathematical operation and a quality metric representing process variation was arbitrarily assigned for each operation. In order to create process paths that would have a different total variation, a dependency was created between the order in which a process task was executed and the variation in the operation. For example, if "Add A" service was executed first it might have a lower process variation than if it was executed last.

After identifying the services needed for the ring creation process, the web service framework was created. For this prototype, XFire, an open source web services API [39] based on the Java programming language, was used. The primary motivation for selecting XFire over other more widely used solutions by IBM, Microsoft, or Sun Microsystems was the ease of creating services for automatic binding and invocation. Most commercial solutions for developing a web service framework require that the programmer have some knowledge of the specific services that will be consumed. For each service in the system a custom function call or other individualized method for service consumption must be developed prior to run time. Without a more generic method for consuming a web service, a client (the execution agents in the system) will not be able to dynamically adapt to changes in process workflow or available services without reprogramming the web service-client interface.

XFire, unlike other solutions, has a simple, generic interface for consuming web services. Clients developed using this API use a single function that can be executed with input and output parameters not specific to any one web service. An XFire web service can provide the function parameters as part of a semantic description allowing the client to use the same interface for all web services in the system. Because the XFire client uses a generic method for consuming web services, any changes to process workflows or individual services can be incorporated during run time. No human input is required and the system can adapt to changes of this type automatically. In addition to providing a flexible interface for web service consumption, XFire also allows for web services to be created from basic Java objects. This reduces much of the initial complexity associated with developing a specific web service implementation, which is often a major hurdle in creating web service applications.

In order to facilitate the population of the web service registry, each service implemented a number of reporting methods that could be called by the agent framework to glean the additional information necessary for workflow creation and dynamic execution of the services. Because this approach represents the web service details as executable reporting methods rather than through a more descriptive semantic web service interface, it was necessary to create registry management agents that would update the web service registry in two separate steps. First the agent must search the web services deployment file system for the "services.xml" file that is created as part of XFire development. From this file the names and locations of all available services is registered. Once this information has been determined the agent is then able to call the standard reporting methods (which might be unique to each individual system) to update the registered services with the remaining descriptive information. Figures 5.2 and 5.3 show the MA adding the available web services and updating their descriptive information in this two-step process.


Servlet WSRegistry - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Done

Servlet WSRegistry at /APDLRegistry

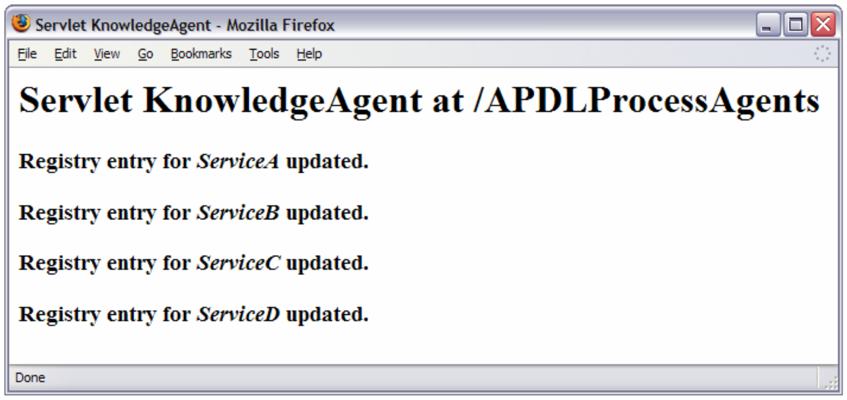
ServiceA added to the registry.

ServiceB added to the registry.

ServiceC added to the registry.

ServiceD added to the registry.

Figure 5.2 Registered web services from "services.xml" file


Servlet KnowledgeAgent - Mozilla Firefox
File Edit View Go Bookmarks Tools Help

Servlet KnowledgeAgent at /APDLProcessAgents

Registry entry for *ServiceA* updated.

Registry entry for *ServiceB* updated.

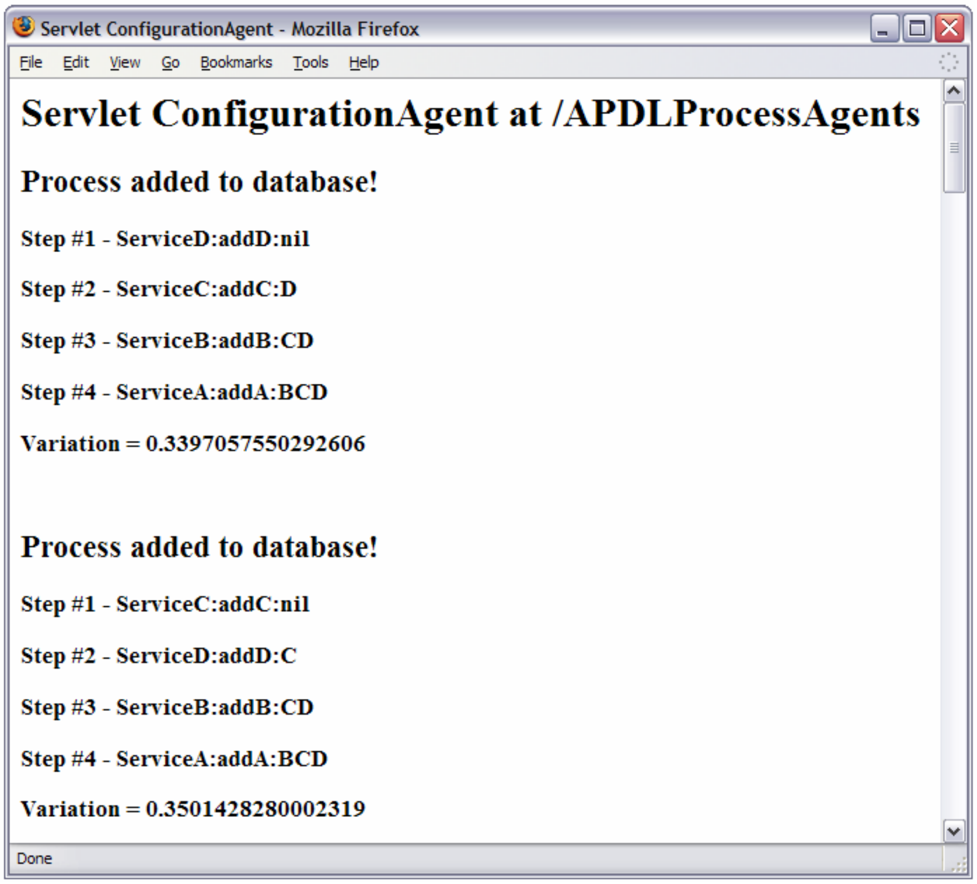
Registry entry for *ServiceC* updated.

Registry entry for *ServiceD* updated.

Figure 5.3 Registry update from web service reporting methods

In addition to the reporting methods, each web service also implements a unique method representing the core task operation. Because each service operation can be used in a number of possible ways (i.e. add A as the initial element, add A to B only, add A to D only, add A to BC, add A to CD, add A to BCD), each service reports correlating sets of input requirements, output produced, and variability for different operation use cases. The input and output sets were defined according to a simple system language that would allow inputs and outputs from different services to be matched during workflow mapping.

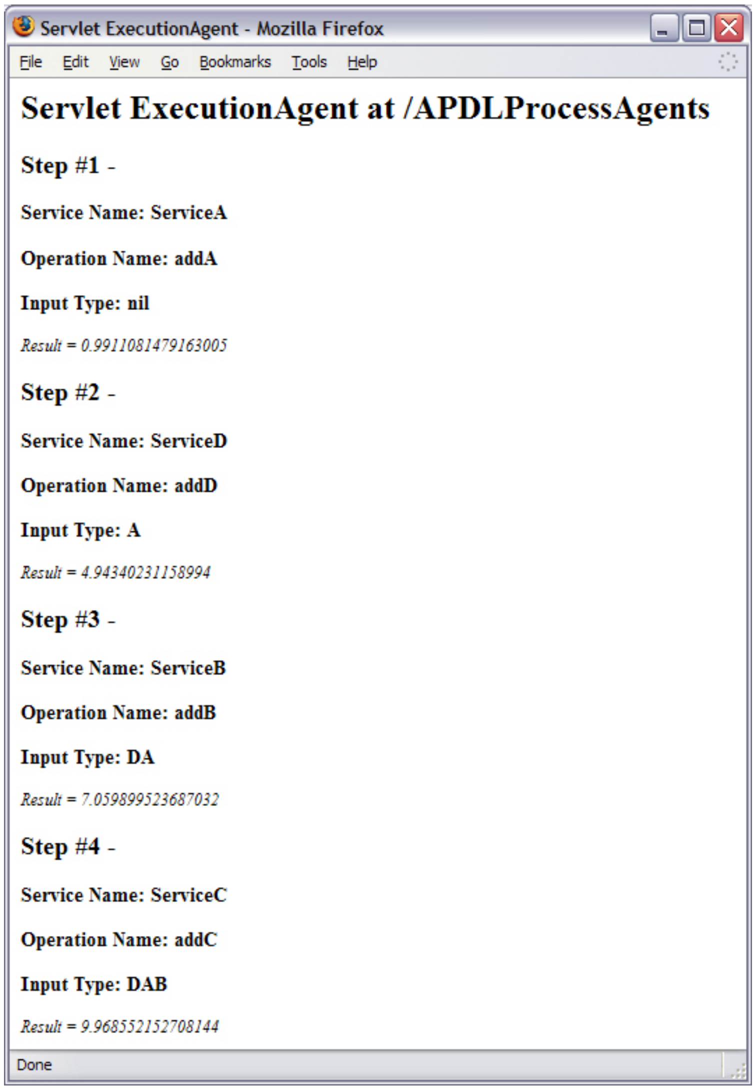
Once the process services were created and registered the CA was able to map all sixteen available processes, two of which are shown in Figure 5.4. Each of these workflow paths was stored in the workflow repository along with the total variation of each process. The EA was implemented so as to select the process with the lowest total variation. Figure 5.5 shows the successful execution of the optimal workflow path based on this criterion.



Servlet ConfigurationAgent - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Done

```
Servlet ConfigurationAgent at /APDLProcessAgents

Process added to database!

Step #1 - ServiceD:addD:nil

Step #2 - ServiceC:addC:D

Step #3 - ServiceB:addB:CD

Step #4 - ServiceA:addA:BCD

Variation = 0.3397057550292606


Process added to database!

Step #1 - ServiceC:addC:nil

Step #2 - ServiceD:addD:C

Step #3 - ServiceB:addB:CD

Step #4 - ServiceA:addA:BCD

Variation = 0.3501428280002319
```

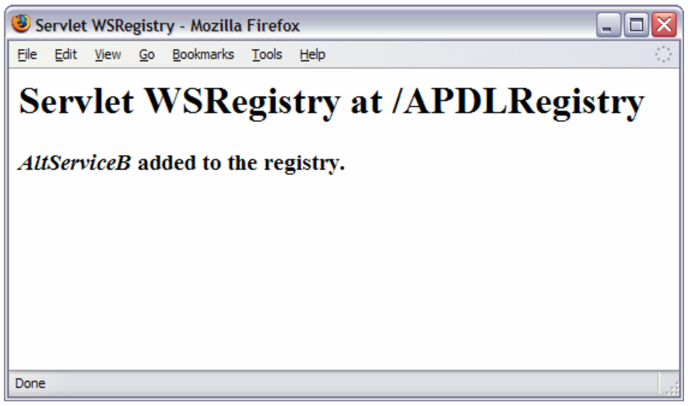
Figure 5.4 Example workflow paths mapped by configuration agent


Servlet ExecutionAgent - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Done

```
Servlet ExecutionAgent at /APDLProcessAgents

Step #1 -
Service Name: ServiceA
Operation Name: addA
Input Type: nil
Result = 0.9911081479163005

Step #2 -
Service Name: ServiceD
Operation Name: addD
Input Type: A
Result = 4.94340231158994

Step #3 -
Service Name: ServiceB
Operation Name: addB
Input Type: DA
Result = 7.059899523687032

Step #4 -
Service Name: ServiceC
Operation Name: addC
Input Type: DAB
Result = 9.968552152708144
```

Figure 5.5 Workflow execution by process execution agent

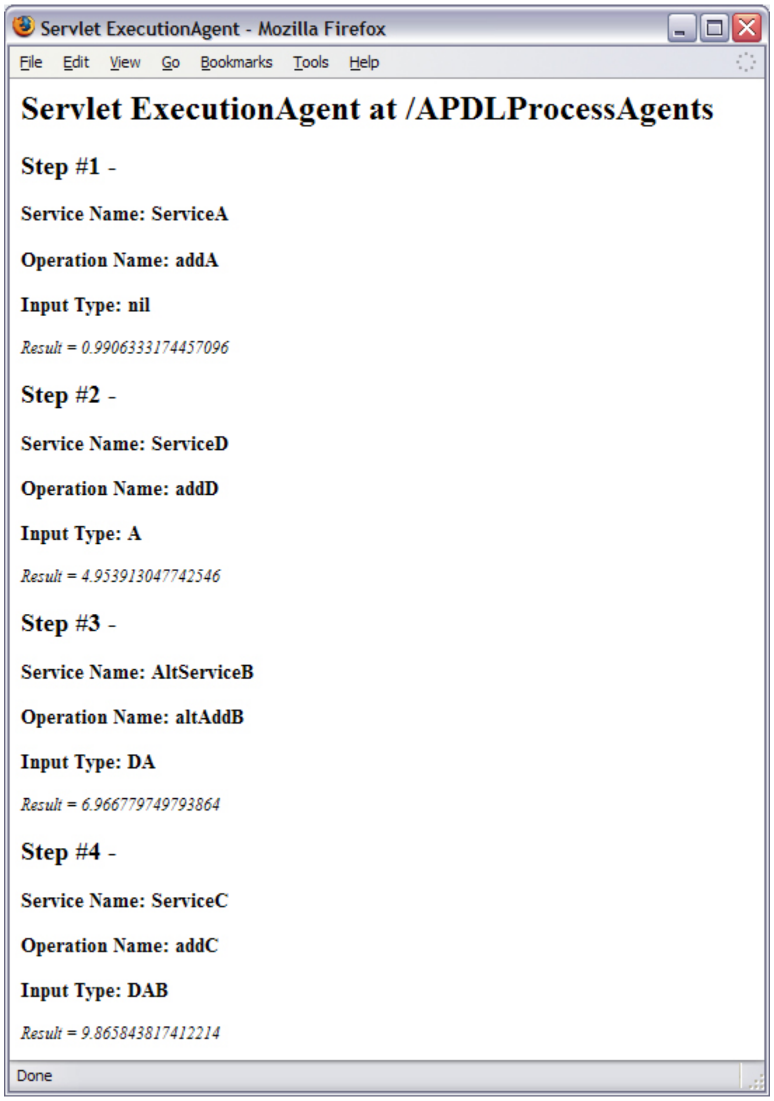
In order to demonstrate the dynamic nature of the system framework, an alternative process task was created that would perform the same operations as the "Add B" web service using a different method which lowered variability. As a result, all workflow paths that included this new service would have a lower total variability than those using the original "Add B" service. Creation and registration of the alternative service required only that a new Java object be created from the same template as the other services and that the agent framework recognize the new service, add it to the registry, and map any new workflow paths. After the web service registry was updated the CA was able to find the additional 16 workflow paths and the new optimal path which included the alternative service was selected and executed by the EA. The results from this procedure can be seen in Figures 5.6 and 5.7.


Servlet WSRegistry - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Done

Servlet WSRegistry at /APDLRegistry

AltServiceB added to the registry.

Figure 5.6 Registry update with alternative web service


Servlet ExecutionAgent - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Done

```
Servlet ExecutionAgent at /APDLProcessAgents

Step #1 -
Service Name: ServiceA
Operation Name: addA
Input Type: nil
Result = 0.9906333174457096

Step #2 -
Service Name: ServiceD
Operation Name: addD
Input Type: A
Result = 4.953913047742546

Step #3 -
Service Name: AltServiceB
Operation Name: altAddB
Input Type: DA
Result = 6.966779749793864

Step #4 -
Service Name: ServiceC
Operation Name: addC
Input Type: DAB
Result = 9.865843817412214
```

Figure 5.7 Workflow execution with alternative web service included

5.2 Case Study #2 – Impeller Example

In order to demonstrate the application of the proposed method to a practical engineering problem, an automated modeling and analysis process for an impeller design was used for this case study. Although the core operations for the necessary web services used in this example required more sophistication and interaction with external applications, the method for creating the services and deploying them in the agent-based framework remained the same.

The first step in creating the impeller design framework was to identify the individual process tasks using a backwards mapping technique. It is important that these tasks are first defined on a general level and are not coupled to any specific CAx tools. This enables a necessarily flexible process that is driven by proven engineering design practices rather than the current implementation of available design tools. For this case study the workflow was subdivided into the following tasks:

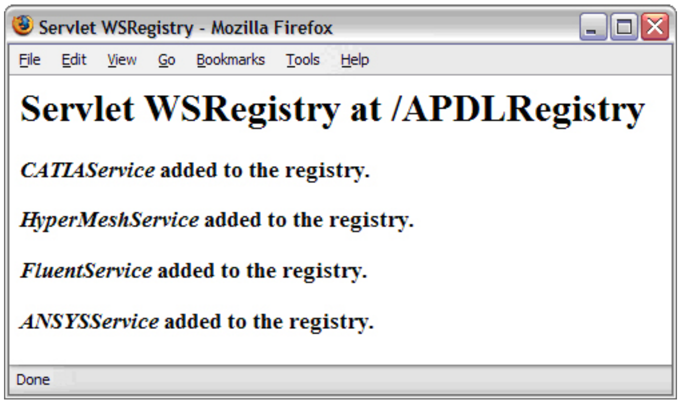
1. Update the parametric models for structural and air solid wedges.
2. Create surface and volume meshes for the air solid wedge.
3. Determine surface pressure values for the air solid wedge.
4. Create surface and volume meshes for the structural wedge.
5. Determine maximum stress values for the structural wedge.

Once this general workflow was identified, specific CAx tools and implementation methods were selected for each process task and their function(s) embedded in an XFire web service. In this example, the process task modules and the resulting web service implementation were far more sophisticated than the ring example. While successful automation of these process tasks required a programmatic interface with parametric CAD models as well as parametric finite-element models for both CFD and stress calculations, the workflow was much less sophisticated than the ring example since it was a simple linearly sequential process. Most practical engineering processes follow a quasi-linear sequential process and are made up of sophisticated services.

In addition to implementing the process design tasks as web services, a system process language was identified from the inputs and outputs of the individual modules. Because the process language followed the same patterns used in the ring example no additional changes were required to the configuration agent. Changes were made, however, to the execution agent. Because there was not a measurable metric for each process module the variation calculations were removed from the execution agent.

Like the ring example, the web service registry was populated by first identifying available modules from the "services.xml" file and then executing predefined reporting methods to provide the necessary execution details and semantic information for each process task (e.g. inputs, outputs, network location). For this case study Derby, an open source database based on Java, JDBC, and SQL standards, was used to store the agent information. A separate instance of the Derby database was also used to store the single linear workflow identified by the CA. Figure 5.8 shows the MA adding the process tasks to this database and Figure 5.9 shows the linear process mapped by the CA.

In order to maintain flexibility, the only system level information required by the individual web services is the name and location of a user specific working directory. Web services use this directory as a repository for inputs, outputs, and any resulting design artifacts, eliminating any need for communication between the services. The execution agent's responsibility is to provide the location of the working directory to all web services during execution.


Servlet WSRegistry - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Done

Servlet WSRegistry at /APDLRegistry

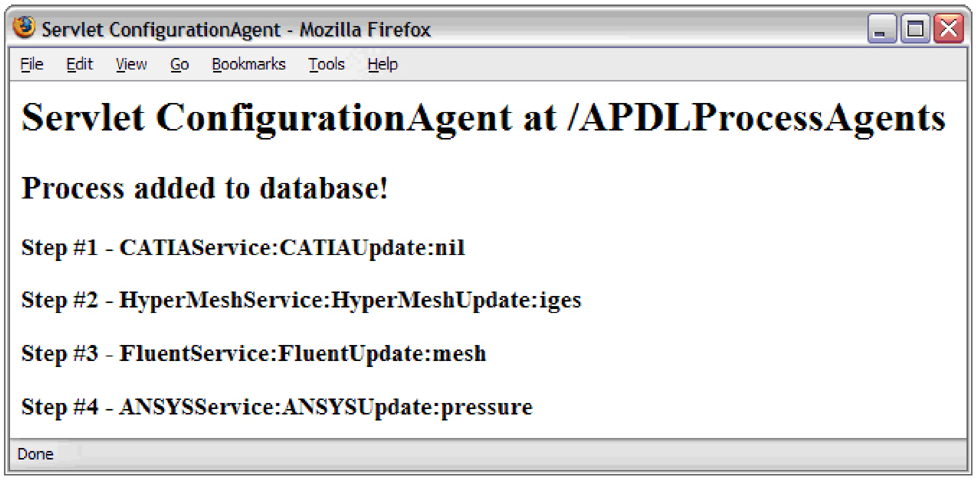
CATIAService added to the registry.

HyperMeshService added to the registry.

FluentService added to the registry.

ANSYSService added to the registry.

Figure 5.8 Registered web services for impeller design process


Servlet ConfigurationAgent - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Done

Servlet ConfigurationAgent at /APDLProcessAgents

Process added to database!

Step #1 - CATIAService:CATIAUpdate:nil

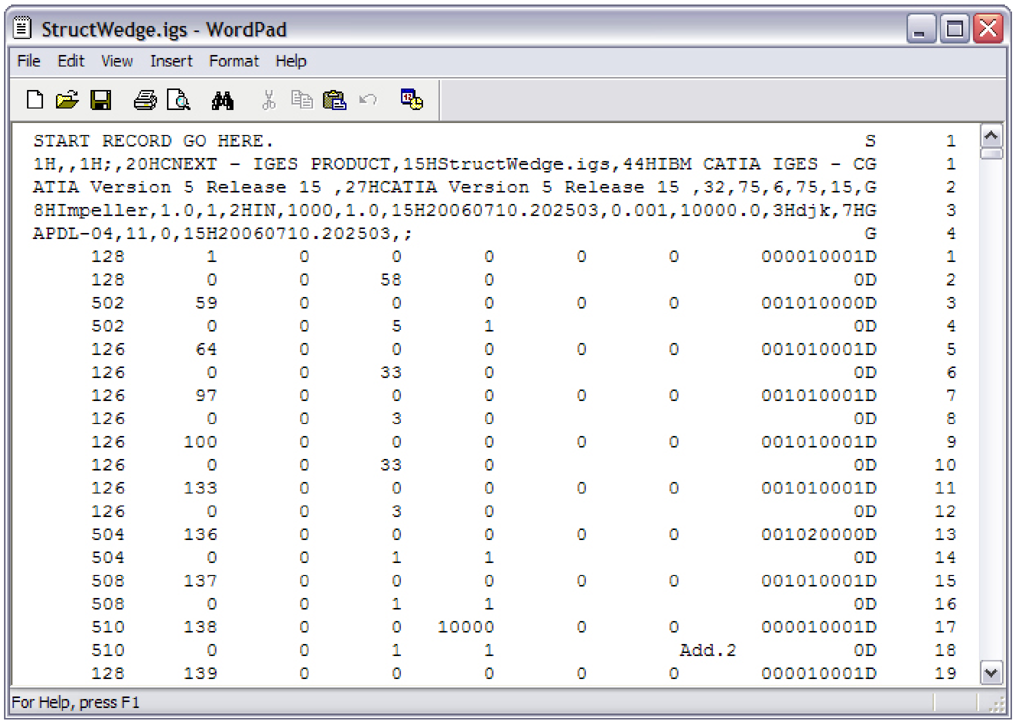
Step #2 - HyperMeshService:HyperMeshUpdate:iges

Step #3 - FluentService:FluentUpdate:mesh

Step #4 - ANSYSService:ANSYSUpdate:pressure

Figure 5.9 Impeller design workflow path mapped by the configuration agent

For this case study the EA is represented by a Java servlet that receives the name and location of the working directory, impeller blade angles, and number of impeller blades as user input. Following the linear process identified by the CA and stored in the Derby database, the execution agent invokes the CAD service which uses the user input parameters to update a parametric model of the impeller in CATIA V5. Once the model has been updated, a structural wedge and air solid wedge are created and saved as IGES files in the working directory. The IGES file is a neutral data format that enables the geometric definition of the impeller to be interpreted by a wide range of CAx tools. Figure 5.10 shows the actual IGES data for the structural wedge while Figure 5.11 shows the CATIA V5 interpretation of this information



```
START RECORD GO HERE.                                                   S      1
1H,,1H;,20HCNEXT - IGES PRODUCT,15HStructWedge.igs,44HIBM CATIA IGES - CG      1
ATIA Version 5 Release 15 ,27HCATIA Version 5 Release 15 ,32,75,6,75,15,G      2
8HImpeller,1.0,1,2HIN,1000,1.0,15H20060710.202503,0.001,10000.0,3Hdjk,7HG      3
APDL-04,11,0,15H20060710.202503,;                                       G      4
     128       1       0       0       0       0       0       000010001D      1
     128       0       0      58       0                               0D      2
     502      59       0       0       0       0       0       001010000D      3
     502       0       0       5       1                               0D      4
     126      64       0       0       0       0       0       001010001D      5
     126       0       0      33       0                               0D      6
     126      97       0       0       0       0       0       001010001D      7
     126       0       0       3       0                               0D      8
     126     100       0       0       0       0       0       001010001D      9
     126       0       0      33       0                               0D     10
     126     133       0       0       0       0       0       001010001D     11
     126       0       0       3       0                               0D     12
     504     136       0       0       0       0       0       001020000D     13
     504       0       0       1       1                               0D     14
     508     137       0       0       0       0       0       001010001D     15
     508       0       0       1       1                               0D     16
     510     138       0       0   10000       0       0       000010001D     17
     510       0       0       1       1                Add.2          0D     18
     128     139       0       0       0       0       0       000010001D     19
```

Figure 5.10 IGES data for the impeller structural wedge

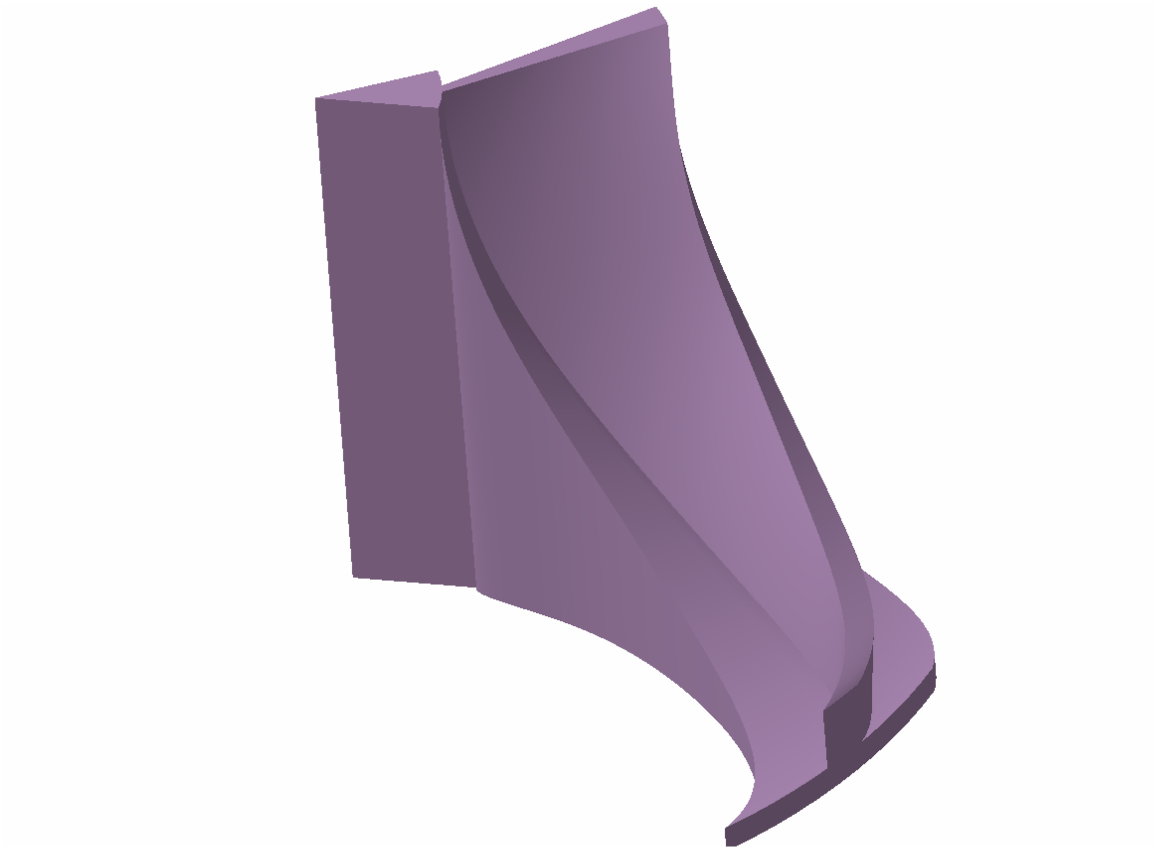


Figure 5.11 CATIA representation of the structural wedge

After receiving confirmation from the CAD service that it has finished executing and successfully created the IGES files for the impeller wedges, the EA invokes the mesh service. This service uses HyperMesh to build surface and volume meshes for finite-element calculations from the air solid IGES file stored in the working directory. Once created, the meshes are then saved to this same directory. Figure 5.12 shows the surface mesh for the air solid wedge created by HyperMesh.

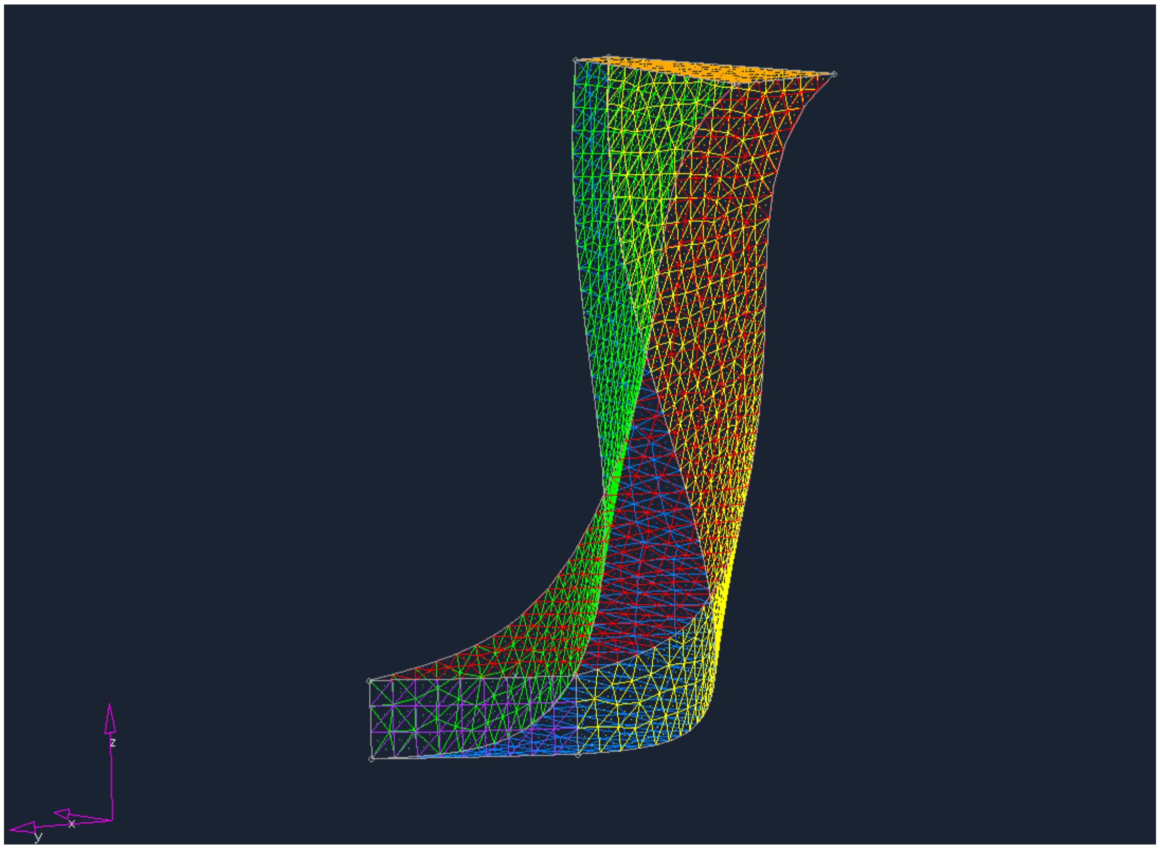


Figure 5.12 Surface mesh for the air solid wedge produced by HyperMesh

The impeller design process continues after the mesh service has executed, stored the output in the working directory, and returned control to the EA. The next step identified by the CA is providing pressure values from a fluid analysis package such as Fluent. The EA accomplishes this by invoking the pressure service with the working directory as input. Fluent uses the HyperMesh output files and iterates through air flow calculations to determine the pressure on the impeller blade surfaces. The final solution is then written to a text file and saved in the working directory. Figure 5.13 is a screenshot of the pressure distribution that is also output to the working directory by this service.

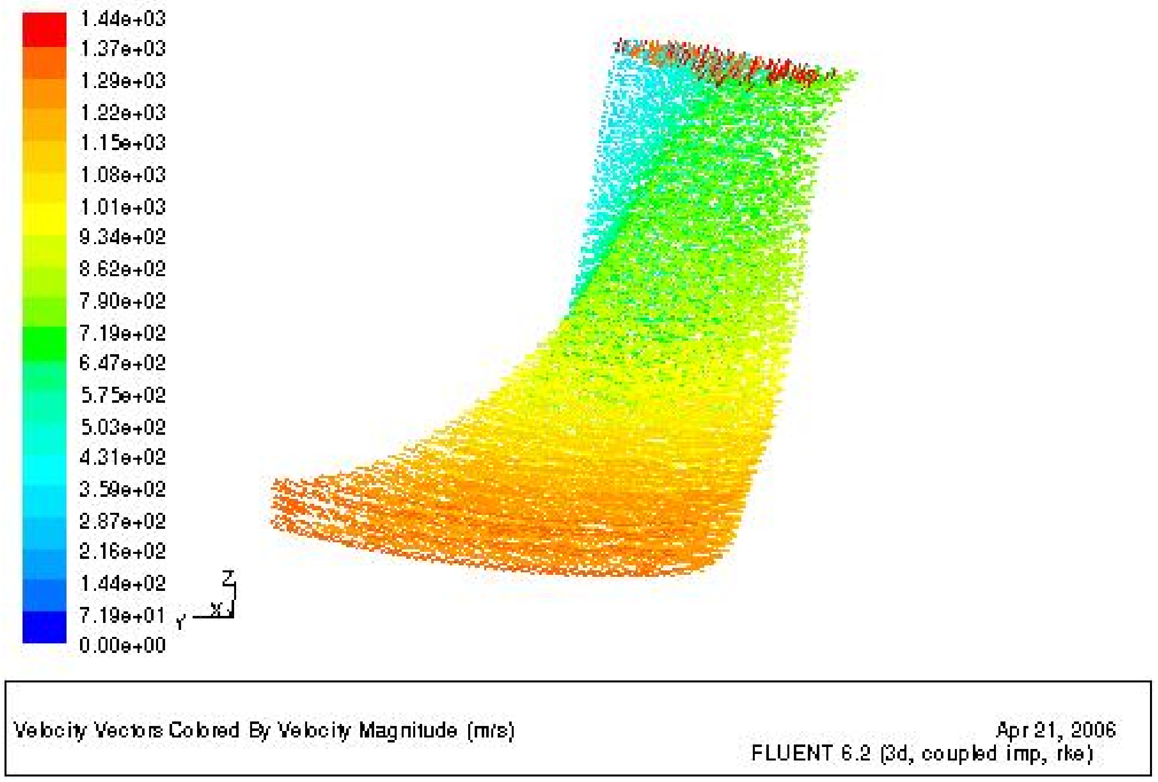


Figure 5.13 Fluent pressure distribution for the air solid wedge

Execution of the stress analysis service follows a procedure very similar to the operation of the pressure service. The stress analysis service both writes and runs an ANSYS macro that inputs the pressure data from the working directory to produce stress values for the structural wedge. Figure 5.14 shows the Von Mises stress plot that is part of the service output.

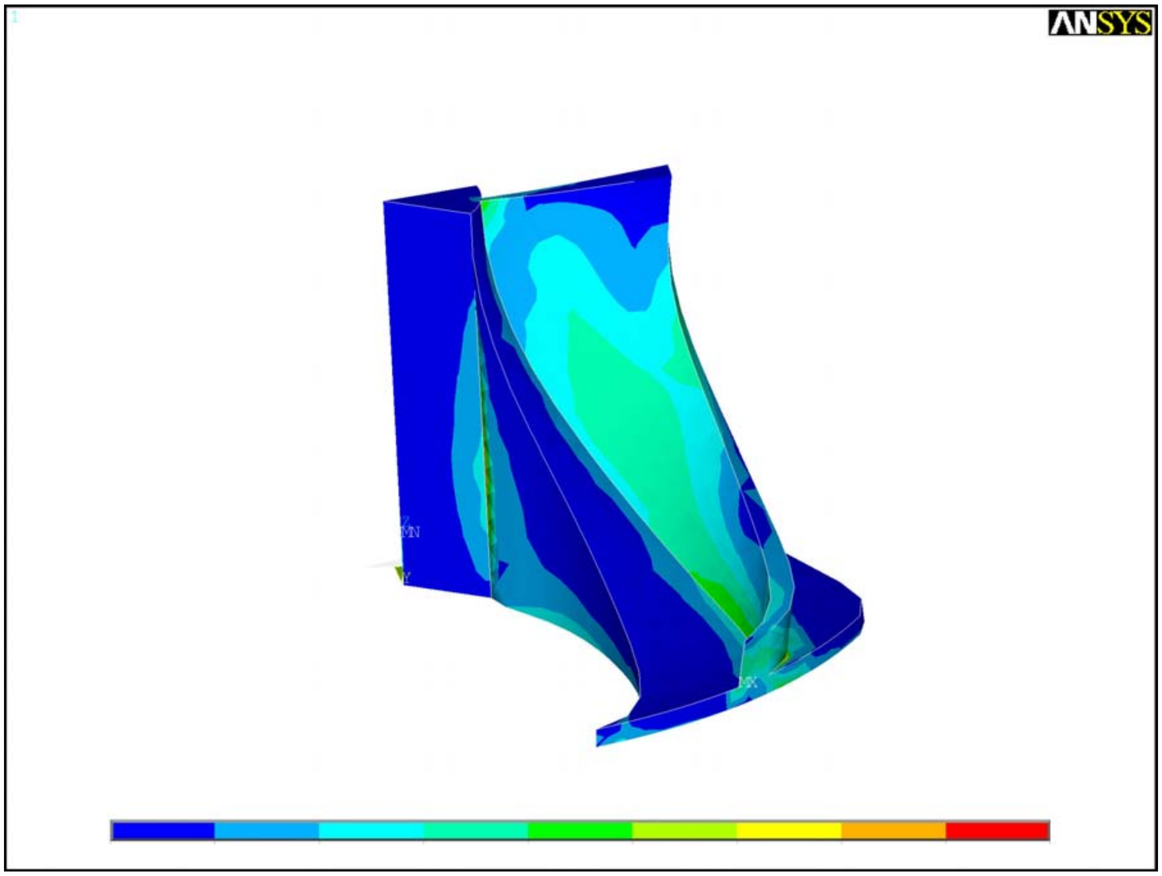


Figure 5.14 Stress plot produced by ANSYS for the structural wedge

It is important to note that because this design process is composed of self contained services integrated into a flexible framework introduction of a new CAx tool or moving an existing tool would only require that the registry be reinitialized. The dynamic nature of the framework will then create a new workflow using the new web service and add this to the database where it can be accessed for execution by the EA. Sophisticated workflow paths that include bifurcation, loops, etc. can be handled using process execution languages such as BPEL or by introducing recursion and hierarchy into the service definitions [6] without changing the methods already presented.

CHAPTER 6 CONCLUSION

The presented method of establishing a dynamic system framework that divides the software maintenance into development of a generic system framework and specific process task modules allows companies to better understand and manage custom software investment. As a result, a company can implement a management strategy to maintain a system framework for all automated engineering projects and then provide templates for process task module development. The web services can be created and managed by the engineers involved with the specific process. Typically they are the most knowledgeable about the specific task and the tools used to accomplish it and therefore are the most qualified to create and manage the specific process task module. Any requirement to know the system-level protocols for tying task modules together into automated process systems is eliminated.

The method presented therefore provides a way to better match the normal decomposition of knowledge and tasks based on personnel involved in the process and a more manageable approach to software development and maintenance. This will hopefully eliminate any resistance to engage in process automation projects require to truly implement mass customization.

The goals outlined in the introduction of this work were met during the course of this research. First, a flexible design system was defined using web service and agent technologies. Specifically, the definition of a Web Service Registry Agent, Workflow Configuration Agent, and Workflow Execution Agent was developed along with a framework for integrating these agents. A process for identifying and creating web services from a specific process was also presented. Finally, the feasibility of the proposed system was demonstrated with a generic example (the ring structure) and a specific engineering example (the impeller design).

Future work includes the implementation of hierarchy, recursion, and other sophisticated methods to allow bifurcation and looping in processes. More sophisticated implementation of EAs to accomplish multi-disciplinary optimization, design studies, etc. is also needed. In addition, future research should include developing methods to analyze and contrast competing workflow paths to develop improved efficiency and scheduling.

CHAPTER 7 REFERENCES

[1] Tseng, Mitchell M. and Jianxin Jiao. "Mass Customization." In *Handbook of Industrial Engineering: Technology and Operations Management, 3rd Edition*, ed. Gravriel Slavendy, 685. Hoboken, NJ: John Wiley & Sons, Inc., 2001.

[2] Pine, Joseph B. *Mass Customization: The New Frontier in Business Competition*. Boston: Harvard Business School Press, 1993.

[3] Lander, Susan E. "Issues in Multi-agent Design Systems." *IEE Expert*, 12, no. 2 (1997): 18-26.

[4] Roach, Gregory M., Cox, Jordan J., and Carl D. Sorenson. "The Product Design Generator: A System for Producing Design Variants." *International Journal of Mass Customization*, 12, no. 1: 2005.

[5] Roach, Gregory M. "The Product Design Generator – A Next Generation Approach to Detailed Design." Ph.D. diss., Brigham Young University, 2003.

[6] Young, Jared M. "Nesting Automated Design Modules in an Interconnected Framework." M.S. thesis, Brigham Young University, 2005.

[7] Wooldridge, Michael and Nicolas R. Jennings. "Pitfalls of Agent-oriented Development." In *Proceedings of the Second International Conference on Autonomous Agents held in Minneapolis, MN 10-13 May 1998*, 385-391.

[8] Woofinden, M. J. "Quantifying Reuse in Computer Models." M.S. thesis, Brigham Young University, 2003.

[9] Feng, Shaw C. "Preliminary Design and Manufacturing Planning Integration Using Web-based Intelligent Agents." *Journal of Intelligent Manufacturing* 16 (2005): 423-437.

[10] Shen, Weiming, Norrie, Douglas H. and Jean-Paul A. Barthès. *Mult-agent Systems for Concurrent Intelligent Desing and Manufacturing*. London, England: Taylor and Francis, 2001.

[11] Darr, Timothy P. and William P. Birmingham. “Automated Design for Concurrent Engineering. “ *IEEE Expert* 9 (1999), no. 5: 35-42.

[12] Campbell, Matthew I., Cagan, Jonathan, and Kenneth Kotovsky. “A-Design: An Agent-based Approach to Conceptual Design in a Dynamic Environment.” *Research in Engineering Design* 11 (1999): 172-192.

[13] Campbell, Matthew I., Cagan, Jonathan, and Kenneth Kotobsky. “The A-Design Approach to Managing Automated Design Synthesis.” *Research in Engineering Design* 14 (2003): 12-24.

[14] Rosenman, Mike and Fujun Wang. “CADOM: A Component Agent-based Design-Oriented Model for Collaborative Design.” *Research in Engineering Design* 11 (1999): 193-205.

[15] Parunak, H.V.D., Ward, A., Fleischer, M. and J. Sauter. “A Marketplace of Design Agents for Distributed Concurrent Set-based Design.” In *ISPE/CE97: Fourth ISPE International Conference on Concurrent Engineering: Research and Applications held in Troy, MI 20-22 August 1997.*

[16] Sun Developer Network. “Service-Oriented Architecture and Web Services: Concepts, Technologies, and Tools.” http://java.sun.com/developer/technicalArticles/WebServices/soa2/index.html (valid as of Nov. 2006).

[17] Web Services Interoperability Organization. “WS-I Basic Profile Version 1.1.” http://www.ws-i.org/Profiles/BasicProfile-1.1-2004-08-24.html (valid as of Nov. 2006).

[18] W3C. “Extensible Markup Language (XML) 1.1.” http://www.w3.org/TR/xml11/ (valid as of Nov. 2006).

[19] W3C. “SOAP Version 1.2 Part 0: Primer.” http://www.w3.org/TR/soap12-part0/ (valid as of Nov. 2006)

[20] Wooldridge, Michael. “Intelligent Agents” In *Multi-agent Systems*, ed. G. Weiss. Boston: The MIT Press, 1999.

[21] Wooldridge, Michael and Ian Dickinson. “Agents are not (Just) Web Services: Considering BDI Agents and Web Services.” In *Proceeding of the 2005 Workshop on Service-oriented Computing and Agent-based Engineering (SOCABE 2005) held in The Hague, Netherlands, Jul 2005.*

[22] Wooldridge, Michael and Nicholas R. Jennings. ”Intelligent Agents: Theory and Practice.” *Knowledge Engineering Review*, 10, no. 2 (1995).

[23] Russell S. and P. Norvig. *Artifical Intelligence: A Modern Approach*. Prentice-Hall, 1995.

[24] Foundation for Intelligent Physical Agents. "FIPA ACL Message Structure Specification." http://www.fipa.org/specs/fipa00061/index.html (valid as of Nov. 2006).

[25] W3C Web Service Architecture Working Group. " Web Service Architecture." http://www.w3.org/TR/ws-arch (valid as of Nov. 2006).

[26] Greenwood, Dominic and Monique Calisti. "Engineering Web Service-Agent Integration" *In IEEE Systems, Cybernetics and Man Conference, The Hague, Netherlands, October 2004.*

[27] Java Agent DEvelopment Framework. "An Open Source Platform for Peer-to-Peer Agent Based Applications." http://jade.tilab.com/ (valid as of Nov. 2006).

[28] The Foundation for Intelligent Physical Agents. "Welcome to FIPA!" http://www.fipa.org/ (valid as of Nov. 2006).

[29] Maximilien, E. M. and M.P. Singh. "Agent-based Architecture for Autonomic Web Service Selection." In *Proc. of the 1st International Workshop on Web Services and Agent Based Engineering, Sydney, Australia, July 2003.*

[30] Buhler, P., Vidal, J. N. and H. Verhagen. "Adaptive Workflow = Web Services + Agents." In *Proc. of the International Conference on Web Services, Las Vegas, NV July 2003.* 131-17. CSREA Press.

[31] Laukkanen, M. and H. Helin. "Composing Workflows of Semantic Web Services." In *Proc. of the 1st International Workshop on Web Services and Agent Based Engineering, Sydney, Australia, July 2003.*

[32] Sycara, K., Paolucci, M., Soudry, J., and N. Srinivasan. "Dynamic Discovery and Coordination of Agent Based Semantic Web Services." *IEEE Internet Computing*. May-Jun, 2004.

[33] McIlraith, S. A., Son, T.C. and Zeng Honglei. "Semantic Web Services." *IEEE Intelligent Systems*. Mar-Apr, 2001.

[34] Hendler, J. "Agents and the Semantic Web." *IEEE Intelligent Systems*. Mar-Apr, 2001.

[35] IBM. "Web Services by IBM: UDDI." http://www-306.ibm.com/software/solutions/webservices/uddi/shutdown_faq.html (valid as of Nov. 2006).

[36] W3C. “OWL-S: Semantic Markup for Web Services.” http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/ (valid as of Nov. 2006).

[37] W3C. “Semantic Annotations for WSDL.” http://www.w3.org/TR/sawsdl/ (valid as of Nov. 2006)

[38] IBM Developer Works. “Business Process Execution Language for Web Services.” ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf (valid as of Nov. 2006)

[39] Codehaus XFire. “Codehaus XFire.” http://xfire.codehaus.org/ (valid as of Nov. 2006)

[40] Baker, Albert D., Parunak, Van Dyke H., and Kutluhan Erol. “Agents and the Internet: Infrastructure for Mass Customization.” *IEE Internet Computing*. 1999.

[41] Sun, J., Zhang, Y. F. , and A. Y. C. Nee. “A Distributed Multi-agent Environment for Product Design and Manufacturing Planning.” *International Journal of Production Research*, 39, no. 4 (2001), 625-645.

[42] Shen, Weiming, and Ghenniwa. “A Distributed Multidisciplinary Design Optimization Framework Based on Web and Agents.” In *Proceedings of the 2002 Design Engineering Technical Conferences, Montreal, Canada, DETC2002/CIE-34461.*

[43] Wang, S.L, Xia, H., Liu, F., Tao, G.B., and Z. Zhang. “Agent-Based Modeling and Mapping of a Manufacturing System.” *Journal of Materials Processing Technology*. 129 (2002): 518-523.

[44] Anosike, Anthony and David Zhang. “An Agent-Oriented Modelling Approach for Agile Manufacturing.” School of Engineering and Computer Science, University of Exeter, Exeter, United Kingdom, 2000.

[45] Shen and Weiming. “Distributed Manufacturing Scheduling Using Intelligent Agents.” *IEE Intelligent Systems*. Jan-Feb 2002, 88 - 94.

[46] Skolicki, Zbigniew and Tomasz Arciszewski. “Intelligent Agents in Design.” In *Design Engineering Technical Conferences, Chicago, Illinois, USA, 2003.* DETC2003/DTM-48671.

[47] Zha, Xuan F., Li, Ling L., and Samuel Y. E. Lim. “A Multi-Agent Intelligent Environment For Rapid Assembly Design.” In *Planning and Simulation, Design Engineering Technical Conferences, Salt Lake City, UT, USA, 2004.* DETC2004-57713.

[48] Nanda, Jyotirmaya, Thevenot, Henri J., Simpson, Timothy W., Kumara, and R.T. Soundar. “Exploring Semantic Web Technologies for Product Family Modeling.” In *Design Engineering Technology Conferences, Salt Lake City, UT, 2004.* DETC2004-57683.

[49] Hao, Qi, Shen, Weiming, Zhang, Zhan, Park, Seong-Whan, Lee, and Jai-Kyung. “A Multi-Agent Framework for Collaborative Engineering Design and Optimization”, *Design Engineering Technical Conferences, Salt Lake City, UT, 2004.* DETC2004-57686.

[50] Knapik, Michael and Jay Johnson. *Developing Intelligent Agents for Distributed Systems*. New York: McGraw-Hill, 1998.

[51] Murch, Richard and Tony Johnson. *Intelligent Software Agents*. Upper Saddle River, NJ: Prentice Hall PTR, 1999.

APPENDIX A GUIDELINES FOR IMPLEMENTING THE SYSTEM FRAMEWORK (README.TXT)

```
1.  Download/Install the following in the folder C:\dev -

       Netbeans 5.0 (http://www.netbeans.info/downloads/download.php?type=5.0)
              - Download the IDE + Application Server Bundle.
              - This will give you an integrated Tomcat server and
                integrated Derby database.
              - There is no need to install a seperate instance of these
                programs.
              - Install Netbeans in C:\dev\netbeans-5.0
              - Install Sun App Server in C:\dev\sun-appserver-8.2

       XFire (http://xfire.codehaus.org/Download)
              - Download the binary distribution in zip package.
              - To install just extract to the C:\dev folder.

       Maven (http://maven.apache.org/download.html)
              - Download the binary distribution in zip package.
              - To install just extract to the C:\dev folder.

       Ring Example (http://www.et.byu.edu/~djk22)
              - After downloading/extracting move the "projects" and
                "data" folders to C:\dev


       ***Note:  If you use something other than C:\dev you will need to make
       some changes in the registry agent.


2.  Set the Maven environment variables:
       - Right click on "My Computer" and select "Properties"
       - Advanced -> Environment Variables
       - To the system path add the maven bin path (i.e. C:\dev\maven-2.0.4\bin)
       - Add a new variable call JAVA_HOME set to the value of the
         Installed jdk (i.e. C:\Program Files\Java\jdk1.5.0_06)


3.  Set the Netbeans parameters for Derby
       - Open Netbeans
       - Tools -> Options -> Advanced Options -> IDE Configuration ->
         Server and External Tool Settings -> Derby Database
       - Make sure that the derby location is correct
         (i.e. C:\dev\sun-appserver-8.2\derby)
       - Set the database location to C:\dev\data
```

```
4.  Start the database
       - Tools -> Derby Database -> Start Derby Server


5.  Fix any driver errors
       - Expand "Databases" in the runtime tree
       - Expand "Drivers"
       - Fix any drivers in the list with a red cross
              - Right click on the driver and select "Customize"
              - Add the correct path and remove the incorrect path
              - Click on the find button
              - Click on OK


6.  Create a new Derby connection
       - Right click on the "Apache Derby (Net)" driver and
         select "Connect Using..."
       - Change the Database URL to jdbc:derby://localhost:1527/APDLdb
       - Change the User Name to apdl
       - Change the Password to apdl
       - Click on OK
       - Verify connection by expanding the connection and then
         expanding "Tables" to see database tables
       - View table data by right clicking on a table and
         selecting "View Data..."


7.  Copy Derby jar files to Tomcat directory
       - Copy all the jar files in C:\dev\sun-appserver-8.2\derby\lib
       - Paste all the copied files in
         C:\dev\netbeans-5.0\enterprise2\jakarta-tomcat-5.5.9\common\lib


8.  Add a new Tomcat user
       - Open the tomcat-users.xml file located at
         C:\Documents and Settings\UserName\.netbeans\5.0\
         jakarta-tomcat-5.5.9_base\conf
       - Add a new user with your desired username and password
       - Make sure you include both the manager and admin roles


8.  Add the "APDLRegistry" and "APDLProcessAgent" projects
    (NOT THE "APDLServices" PROJECT!).
       - File -> Open Project...


9.  Create a user library for the XFire jars
       - Tools -> Library Manager -> New Library...
       - Add C:\dev\xfire-1.1.2\xfire-all-1.1.2.jar to the new library
       - Add all the jar files found in C:\dev\xfire-1.1.2\lib


10.  Add the new library to the APDLProcessAgents project
       - Right click on project libraries and select "Add Library..."


11.  Start the Tomcat server
       - In the Netbeans runtime tab right click on the Tomcat
         server and select "Start"


12.  Build and deploy projects to the Tomcat server
       - Right click on project and select "Build Project"
       - Right click on project and select "Deploy Project"


13.  Build the web services
       - Browse to the APDLServices project folder and run the maven.bat file
       - After maven has downloaded all the files from the repository a
         new folder named "target" will have been created
       - Make sure that the file APDLServices-1.0.war exists in the
         "target" folder


14.  Deploy the web services
       - Browse to http://localhost:8084
       - Select the Tomcat Manager
       - Input the username and password you created
       - From the "WAR file to deploy" section browse to the war file
         in the "target" folder and deploy the file

15.  Run the agents
       - Browse to http://localhost:8084/APDLRegistry/WSRegistry to call
         the first servlet (this adds the name and wsdl location to
         the database)
       - Browse to http://localhost:8084/APDLProcessAgents/KnowledgeAgent
         to update the registry entries with additional details
       - Browse to
         http://localhost:8084/APDLProcessAgents/
         ConfigurationAgent?endpoint=ABCD to backwards map all the processes
       - Browse to http://localhost:8084/APDLProcessAgents/ExecutionAgent
         to run the best process

***Note:  The alternate B service can be removed by removing its entry
          in the "services.xml" file located at C:\dev\projects\APDLServices\
          src\main\META-INF\xfire.  You will then need to delete the "target"
          folder and rerun the maven batch file.  If you clear the registries
          (run a view data command in Netbeans then replace the text "select *"
          with "delete" and rerun the script) you can rerun the agents to map
          the process without the alternate agent.  No changes need to be made
          to the registries to add a service.
```

APPENDIX B RING EXAMPLE – CODE

Services.xml

```
<beans xmlns="http://xfire.codehaus.org/config/1.0">

  <service>
    <name>ServiceA</name>
    <namespace>APDLServices-1.0/services</namespace>
    <serviceClass>org.apdl.services.ServiceASEI</serviceClass>
    <implementationClass>org.apdl.services.ServiceAImpl</implementationClass>
  </service>

  <service>
    <name>ServiceB</name>
    <namespace>APDLServices-1.0/services</namespace>
    <serviceClass>org.apdl.services.ServiceBSEI</serviceClass>
    <implementationClass>org.apdl.services.ServiceBImpl</implementationClass>
  </service>

  <service>
    <name>ServiceC</name>
    <namespace>APDLServices-1.0/services</namespace>
    <serviceClass>org.apdl.services.ServiceCSEI</serviceClass>
    <implementationClass>org.apdl.services.ServiceCImpl</implementationClass>
  </service>

  <service>
    <name>ServiceD</name>
    <namespace>APDLServices-1.0/services</namespace>
    <serviceClass>org.apdl.services.ServiceDSEI</serviceClass>
    <implementationClass>org.apdl.services.ServiceDImpl</implementationClass>
  </service>

  <service>
    <name>AltServiceB</name>
    <namespace>APDLServices-1.0/services</namespace>
    <serviceClass>org.apdl.services.AltServiceBSEI</serviceClass>
   <implementationClass>org.apdl.services.AltServiceBImpl</implementationClass>
  </service>

</beans>
```

WSRegistry.java

```
/*
 * WSRegistry.java
 *
 * Created on June 6, 2006, 7:26 PM
 */

package org.apdl.servlets;

import java.io.*;

import javax.servlet.*;
import javax.servlet.http.*;

import java.sql.*;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.regex.*;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;

/**
 *
 * @author APDL
 * @version
 */
public class WSRegistry extends HttpServlet {
    /**
     * Parameters for "services.xml" file search
     */
    private PrintWriter out;
    private static String webServicesFile = "services.xml";
    private static String filePath = "C:/dev/projects/APDLServices";
    private boolean fileFound = false;
    private String webServiceDoc = null;

    /** Processes requests for both HTTP <code>GET</code> and <code>POST</code>
methods.
     * @param request servlet request
     * @param response servlet response
     */
    protected void processRequest(HttpServletRequest request,
HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet WSRegistry</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Servlet WSRegistry at " + request.getContextPath () +
"</h1>");

        /**
         * Database connection parameters
         */
        String dbUrl = System.getProperty("APDLdb.url",
"jdbc:derby://localhost:1527/APDLdb");
        String dbDriver = System.getProperty("APDLdb.driver",
"org.apache.derby.jdbc.ClientDriver");
        String dbUser = System.getProperty("APDLdb.user", "apdl");
        String dbPassword = System.getProperty("APDLdb.password", "apdl");

        /**
         * Connect to derby database
         **/
        Connection conn = null;
        try {
            Class.forName(dbDriver).newInstance();
            conn = DriverManager.getConnection(dbUrl, dbUser, dbPassword);
       } catch (Exception e) {
            out.println("<h3>" + e + "</h3>");
            out.println("<h3>Failed to load database driver!</h3>");
       }

        /**
         * Get existing services from database
         **/
        ResultSet rs = null;
        try {
            String sql = "SELECT service_name FROM apdl.web_services";
            PreparedStatement psGet = conn.prepareStatement(sql);
            rs = psGet.executeQuery();
       } catch (Exception e) {
            out.println("<h3>" + e + "</h3>");
            out.println("<h3>Failed to retrieve services from database!</h3>");
       }

        /**
         * Store existing service information in an ArrayList
         **/
        ArrayList<String> existingServices = new ArrayList<String>();
        boolean databaseEmpty = true;
        try {
            while (rs.next()) {
                String service = rs.getString("service_name");
                existingServices.add(service);
                databaseEmpty = false;
            }
            rs.close();
       } catch (Exception e) {
            out.println("<h3>" + e + "</h3>");
            out.println("<h3>Failed to store existing service
information!</h3>");
       }

        /**
         * Find "services.xml" deployment description file
         */
        File dir = new File(filePath);
        try {
            findFile(dir);
        } catch (Exception e) {
            out.println("<h3>" + e + "</h3>");
            out.println("<h3>Failed to find file \"services.xml\"!</h3>");
        }

        Document webServices = null;
        try {
            File servicesXML = new File(webServiceDoc);
            DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
            dbf.setValidating(false);
            dbf.setNamespaceAware(true);
            DocumentBuilder documentBuilder = dbf.newDocumentBuilder();
            webServices = documentBuilder.parse(servicesXML);
        } catch (Exception e) {
            out.println("<h3>" + e + "</h3>");
            out.println("<h3>Failed to parse file \"services.xml\"!</h3>");
        }

        /**
         * Parse web service names
         */
        ArrayList<String> serviceNames = new ArrayList<String>();
        ArrayList<String> serviceNamespaces = new ArrayList<String>();
        NodeList nl = webServices.getElementsByTagName("service");
        for (int n = 0; n < nl.getLength(); n++) {
            Element service = (Element) nl.item(n);
            String name =
service.getElementsByTagName("name").item(0).getFirstChild().getNodeValue();
            String namespace =
service.getElementsByTagName("namespace").item(0).getFirstChild().getNodeValue(
);

            if (databaseEmpty == false) {
                boolean matchFound = false;
                Iterator iter = existingServices.iterator();
                while (matchFound != true && iter.hasNext()) {
                    String existingName = (String)iter.next();
                    if (name.equals(existingName))
                        matchFound = true;
                }
                if (matchFound == false) {
                    serviceNames.add(name);
                    serviceNamespaces.add(namespace);
                }
            }

            if (databaseEmpty == true) {
                serviceNames.add(name);
                serviceNamespaces.add(namespace);
            }
        }

        /**
         * Get url for servlet context
         */
        StringBuffer url = new StringBuffer();
        url.append(request.getScheme() + "://");
        url.append(request.getServerName());
        if(request.getServerPort()!=80)
            url.append(":" + request.getServerPort());
        String uri = url.toString();

        /**
         * Add services to database
         **/
        for (int i=0; i<serviceNames.size(); i++) {
            String serviceName = serviceNames.get(i);
            String serviceNamespace = serviceNamespaces.get(i);
            String wsdlLocation = uri + "/" + serviceNamespace + "/" +
serviceName + "?wsdl";

            try {
                String sql = "INSERT INTO apdl.web_services (service_name,
wsdl_location) VALUES (?, ?)";
                PreparedStatement psSet = conn.prepareStatement(sql);
                psSet.setString(1, serviceName);
                psSet.setString(2, wsdlLocation);
                psSet.execute();
            } catch (Exception e) {
                out.println("<h3>" + e + "</h3>");
                out.println("<h3>Failed to load service(s) into
database!</h3>");
            }
            out.print("<h3><i>" + serviceName + "</i> added to the
registry.</h3>");
        }

        /**
         * Close database connection
         */
        try {
            conn.close();
        } catch (Exception e) {
            out.println("<h3>" + e + "</h3>");
            out.println("<h3>Error during file search!</h3>");
        }

        /**
         * Close HTML output
         */
        out.println("</body>");
        out.println("</html>");
        out.close();
    }

    // <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click
on the + sign on the left to edit the code.">
    /** Handles the HTTP <code>GET</code> method.
     * @param request servlet request
     * @param response servlet response
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

    /** Handles the HTTP <code>POST</code> method.
     * @param request servlet request
     * @param response servlet response
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse
response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

    /** Returns a short description of the servlet.
     */
    public String getServletInfo() {
        return "Short description";
    }
    // </editor-fold>

    /**
     * Search method for finding the file "services.xml"
     */
    protected void findFile(File dir) {
        if (fileFound == false) {
            try {
                if (dir.isDirectory()) {
                    String[] children = dir.list();
                    String filename;
                    if (children != null) {
                        for (int i=0; i<children.length; i++) {
                            filename = children[i];
                            File dirChild = new File(dir, children[i] + "\\");
                            if (dirChild.isDirectory()) {
                                findFile(dirChild);
                            } else {
                                if (filename.equals(webServicesFile)) {
                                    webServiceDoc = dir + "\\" + filename;
                                    fileFound = true;
                                }
                            }
                        }
                    }
                }
            } catch (Exception e) {
                out.println("<h3>" + e + "</h3>");
                out.println("<h3>Failed to close database connection!</h3>");
            }
        }
    }

}
```

KnowledgeAgent.java

```
/*
 * KnowledgeAgent.java
 *
 * Created on June 6, 2006, 11:29 PM
 */

package org.apdl.agents;

import java.io.*;
import java.net.URL;

import javax.servlet.*;
import javax.servlet.http.*;

import java.sql.*;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.regex.Pattern;
import org.codehaus.xfire.client.Client;

/**
 *
 * @author APDL
 * @version
 */
public class KnowledgeAgent extends HttpServlet {

    /** Processes requests for both HTTP <code>GET</code> and <code>POST</code>
methods.
     * @param request servlet request
     * @param response servlet response
     */
    protected void processRequest(HttpServletRequest request,
HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet KnowledgeAgent</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Servlet KnowledgeAgent at " + request.getContextPath
() + "</h1>");

        /**
         * Database connection parameters
         */
        String dbUrl = System.getProperty("APDLdb.url",
"jdbc:derby://localhost:1527/APDLdb");
        String dbDriver = System.getProperty("APDLdb.driver",
"org.apache.derby.jdbc.ClientDriver");
        String dbUser = System.getProperty("APDLdb.user", "apdl");
        String dbPassword = System.getProperty("APDLdb.password", "apdl");

        /**
         * Connect to derby database
         **/
        Connection conn = null;
        try {
            Class.forName(dbDriver).newInstance();
            conn = DriverManager.getConnection(dbUrl, dbUser, dbPassword);
       } catch (Exception e) {
            out.println("<h3>" + e + "</h3>");
            out.println("<h3>Failed to load database driver!</h3>");
       }

        /**
         * Get web service wsdl locations from registry
         **/
        ResultSet rs = null;
        try {
            String sql = "SELECT wsdl_location FROM apdl.web_services";
            PreparedStatement psGet = conn.prepareStatement(sql);
            rs = psGet.executeQuery();
       } catch (Exception e) {
            out.println("<h3>" + e + "</h3>");
            out.println("<h3>Failed to retrieve services from database!</h3>");
       }

        /**
         * Store existing service information in an ArrayList
         **/
        ArrayList<String> serviceInfo = new ArrayList<String>();
        try {
            while (rs.next()) {
                String wsdl = rs.getString("wsdl_location");
                serviceInfo.add(wsdl);
            }
            rs.close();
       } catch (Exception e) {
            out.println("<h3>" + e + "</h3>");
            out.println("<h3>Failed to store existing service
information!</h3>");
       }

        for (Iterator iter = serviceInfo.iterator(); iter.hasNext();) {
            String wsdlLocation = (String)iter.next();
            String serviceName = null;
            String serviceDescription = null;
            String databaseName = null;
            String operationName = null;
            String inputString = null;
            String outputString = null;
            String variationString = null;

            try {
                Client client = new Client(new URL(wsdlLocation));
                Object[] servResults = client.invoke("getServiceName", new
Object[] {});
                serviceName = (String)servResults[0];

                Object[] descResults = client.invoke("getServiceDescription",
new Object[] {});
                serviceDescription = (String)descResults[0];

                Object[] dataResults = client.invoke("getDatabaseName", new
Object[] {});
                databaseName = (String)dataResults[0];
                Object[] operResults = client.invoke("getOperationName", new
Object[] {});
                operationName = (String)operResults[0];

                Object[] inResults = client.invoke("getInputString", new
Object[] {});
                inputString = (String)inResults[0];

                Object[] outResults = client.invoke("getOutputString", new
Object[] {});
                outputString = (String)outResults[0];

                Object[] varResults = client.invoke("getVariationString", new
Object[] {});
                variationString = (String)varResults[0];

            } catch (Exception e) {
             out.println("<h3>" + e + "</h3>");
                out.println("<h3>Failed to access web service reporting
methods!</h3>");
            }

            try {
                String sql = "UPDATE apdl.web_services SET service_name=?,
service_description=?, " +
                             "database_name=?, operation_name=?,
input_string=?, output_string=?, " +
                             "variation_string=? WHERE wsdl_location=?";
                PreparedStatement psSet = conn.prepareStatement(sql);
                psSet.setString(1, serviceName);
                psSet.setString(2, serviceDescription);
                psSet.setString(3, databaseName);
                psSet.setString(4, operationName);
                psSet.setString(5, inputString);
                psSet.setString(6, outputString);
                psSet.setString(7, variationString);
                psSet.setString(8, wsdlLocation);
                psSet.execute();
            } catch (Exception e) {
                out.println("<h3>" + e + "</h3>");
                out.println("<h3>Failed to update web service(s)!</h3>");
            }
            out.print("<h3>Registry entry for <i>" + serviceName + "</i>
updated.</h3>");
        }

        /**
         * Close database connection
         */
        try {
            conn.close();
        } catch (Exception e) {
            out.println("<h3>" + e + "</h3>");
            out.println("<h3>Failed to close database connection!</h3>");
        }

        out.println("</body>");
        out.println("</html>");
        out.close();
    }

    // <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click
on the + sign on the left to edit the code.">
```

```
    /** Handles the HTTP <code>GET</code> method.
     * @param request servlet request
     * @param response servlet response
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

    /** Handles the HTTP <code>POST</code> method.
     * @param request servlet request
     * @param response servlet response
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse
response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

    /** Returns a short description of the servlet.
     */
    public String getServletInfo() {
        return "Short description";
    }
    // </editor-fold>
}
```

ConfigurationAgent.java

```
/*
 * ConfigurationAgent.java
 *
 * Created on June 7, 2006, 12:09 AM
 */

package org.apdl.agents;

import java.io.*;
import java.net.URL;

import javax.servlet.*;
import javax.servlet.http.*;

import java.sql.*;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.regex.Pattern;
import org.codehaus.xfire.client.Client;

/**
 *
 * @author APDL
 * @version
 */
public class ConfigurationAgent extends HttpServlet {

    private Connection conn = null;
    private PrintWriter out;

    private ArrayList<String> servNames;
    private ArrayList<String> operNames;
    private ArrayList<String> inStrings;
    private ArrayList<String> outStrings;
    private ArrayList<String> varStrings;
    private ArrayList<String> existingMaps;

    /** Processes requests for both HTTP <code>GET</code> and <code>POST</code>
methods.
     * @param request servlet request
     * @param response servlet response
     */
    protected void processRequest(HttpServletRequest request,
HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        //PrintWriter out = response.getWriter();
        out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet ConfigurationAgent</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Servlet ConfigurationAgent at " +
request.getContextPath () + "</h1>");

        /**
         * Database connection parameters
         */
        String dbUrl = System.getProperty("APDLdb.url",
"jdbc:derby://localhost:1527/APDLdb");
        String dbDriver = System.getProperty("APDLdb.driver",
"org.apache.derby.jdbc.ClientDriver");
        String dbUser = System.getProperty("APDLdb.user", "apdl");
        String dbPassword = System.getProperty("APDLdb.password", "apdl");

        /**
         * Connect to derby database
         **/
        try {
            Class.forName(dbDriver).newInstance();
            conn = DriverManager.getConnection(dbUrl, dbUser, dbPassword);
       } catch (Exception e) {
            out.println("<h3>" + e + "</h3>");
            out.println("<h3>Failed to load database driver!</h3>");
       }

        /**
         * Get web service information from registry
         **/
        ResultSet rsServ = null;
        try {
            String sql = "SELECT service_name, operation_name, input_string,
output_string, " +
                    "variation_string FROM apdl.web_services";
            PreparedStatement psGet = conn.prepareStatement(sql);
            rsServ = psGet.executeQuery();
       } catch (Exception e) {
            out.println("<h3>" + e + "</h3>");
            out.println("<h3>Failed to retrieve services from database!</h3>");
       }

        /**
         * Store existing service information in an ArrayList
         **/
        servNames = new ArrayList<String>();
        operNames = new ArrayList<String>();
        inStrings = new ArrayList<String>();
        outStrings = new ArrayList<String>();
        varStrings = new ArrayList<String>();
        try {
            while (rsServ.next()) {
                String serv = rsServ.getString("service_name");
                servNames.add(serv);

                /* Check for multiple service operaitions */
                String oper = rsServ.getString("operation_name");
                if (oper.indexOf(';') != -1) {
                    String input = rsServ.getString("input_string");
                    String output = rsServ.getString("output_string");
                    String variation = rsServ.getString("variation_string");
                    Pattern p = Pattern.compile(";");
                    String[] operArray = p.split(oper);
                    String[] inArray = p.split(input);
                    String[] outArray = p.split(output);
                    String[] varArray = p.split(variation);

                    operNames.add(operArray[0]);
                    inStrings.add(inArray[0]);
                    outStrings.add(outArray[0]);
                    varStrings.add(varArray[0]);
                    for (int i=1; i<operArray.length; i++) {
                        servNames.add(serv);
                        operNames.add(operArray[i]);
                        inStrings.add(inArray[i]);
                        outStrings.add(outArray[i]);
                        varStrings.add(varArray[i]);
                    }
                } else {
                    operNames.add(oper);
                    String input = rsServ.getString("input_string");
                    inStrings.add(input);
                    String output = rsServ.getString("output_string");
                    outStrings.add(output);
                    String variation = rsServ.getString("variation_string");
                    varStrings.add(variation);
                }
            }
            rsServ.close();
        } catch (Exception e) {
            out.println("<h3>" + e + "</h3>");
            out.println("<h3>Failed to store existing service
information!</h3>");
        }

        /**
         * Get existing process maps from the database
         */
        ResultSet rsMaps = null;
        try {
            String sql = "SELECT process_string FROM apdl.process_maps";
            PreparedStatement psGet = conn.prepareStatement(sql);
            rsMaps = psGet.executeQuery();
        } catch (Exception e) {
            out.println("<h3>" + e + "</h3>");
            out.println("<h3>Failed to retrieve process maps from
database!</h3>");
        }

        /**
         * Store existing process information in an ArrayList
         **/
        existingMaps = new ArrayList<String>();
        try {
            while (rsMaps.next()) {
                String map = rsMaps.getString("process_string");
                existingMaps.add(map);
            }
            rsMaps.close();
        } catch (Exception e) {
            out.println("<h3>" + e + "</h3>");
            out.println("<h3>Failed to store existing process
information!</h3>");
        }

        /**
         * Map and store process paths
         */
        String endpoint = request.getParameter("endpoint");
        mapProcess("", 0.0, endpoint);

        /**
         * Close database connection
         */
        try {
            conn.close();
        } catch (Exception e) {
            out.println("<h3>" + e + "</h3>");
            out.println("<h3>Failed to close database connection!</h3>");
        }

        out.println("</body>");
        out.println("</html>");
        out.close();
    }

    // <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click
on the + sign on the left to edit the code.">
    /** Handles the HTTP <code>GET</code> method.
     * @param request servlet request
     * @param response servlet response
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

    /** Handles the HTTP <code>POST</code> method.
     * @param request servlet request
     * @param response servlet response
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse
response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

    /** Returns a short description of the servlet.
     */
    public String getServletInfo() {
        return "Short description";
    }
    // </editor-fold>

    /**
     * Method for mapping possible process paths and storing results in a
database
     */
    protected void mapProcess(String mapString, Double varValue, String
findNext) {
        if (findNext.equals("nil")) {
            boolean matchFound = false;
            Iterator iter = existingMaps.iterator();
            while (matchFound != true && iter.hasNext()) {
                String existingMap = (String)iter.next();
                if (mapString.equals(existingMap))
                    matchFound = true;
            }
            if (matchFound == false) {
                try {
                    String sql = "INSERT INTO apdl.process_maps " +
                            "(process_string, process_variation) VALUES (?,
?)";
                    PreparedStatement psSet = conn.prepareStatement(sql);
                    psSet.setString(1, mapString);
                    psSet.setDouble(2, varValue);
                    psSet.execute();
                } catch (Exception e) {
                    out.println("<h3>" + e + "</h3>");
                    out.println("<h3>Failed to load new process into
database!</h3>");
                }

                out.println("<h2>Process added to database!</h2>");
                Pattern pStep = Pattern.compile("[;\\s]+");
                String[] steps = pStep.split(mapString);
                for (int i=0; i<steps.length; i++) {
                    out.println("<h3>Step #" + (i+1) + " - " + steps[i] +
"</h3>");
                }
                out.println("<h3>Variation = " + varValue + "</h3>");
                out.println("</br>");
            }
            return;
        }

        for (int i=0; i<operNames.size(); i++) {
            String name = servNames.get(i);
            String operation = operNames.get(i);
            String tempIn = inStrings.get(i);
            String tempOut = outStrings.get(i);
            String tempVar = varStrings.get(i);

            Pattern p = Pattern.compile("[,\\s]+");
            String[] inputs = p.split(tempIn);
            String[] outputs = p.split(tempOut);
            String[] variations = p.split(tempVar);

            for (int j=0; j<outputs.length; j++) {
                if (outputs[j].equals(findNext)) {
                    String newMap = name + ":" + operation + ":" + inputs[j] +
"; " + mapString;
                    Double newVar = Math.sqrt(Math.pow(varValue,2) +
Math.pow(Double.parseDouble(variations[j]),2));
                    mapProcess(newMap, newVar, inputs[j]);
                }
            }
        }
    }

}
```

ExecutionAgent.java

```
/*
 * ExecutionAgent.java
 *
 * Created on June 7, 2006, 4:35 PM
 */

package org.apdl.agents;

import java.io.*;
import java.net.*;

import javax.servlet.*;
import javax.servlet.http.*;

import java.sql.*;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.regex.Pattern;
import org.codehaus.xfire.client.Client;

/**
 *
 * @author APDL
 * @version
 */
public class ExecutionAgent extends HttpServlet {

    /** Processes requests for both HTTP <code>GET</code> and <code>POST</code>
methods.
     * @param request servlet request
     * @param response servlet response
     */
    protected void processRequest(HttpServletRequest request,
HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet ExecutionAgent</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Servlet ExecutionAgent at " + request.getContextPath
() + "</h1>");

        /**
         * Database connection parameters
         */
        String dbUrl = System.getProperty("APDLdb.url",
"jdbc:derby://localhost:1527/APDLdb");
        String dbDriver = System.getProperty("APDLdb.driver",
"org.apache.derby.jdbc.ClientDriver");
        String dbUser = System.getProperty("APDLdb.user", "apdl");
        String dbPassword = System.getProperty("APDLdb.password", "apdl");

        /**
         * Connect to derby database
         **/
        Connection conn = null;
        try {
            Class.forName(dbDriver).newInstance();
            conn = DriverManager.getConnection(dbUrl, dbUser, dbPassword);
       } catch (Exception e) {
            out.println("<h3>" + e + "</h3>");
            out.println("<h3>Failed to load database driver!</h3>");
       }

        /**
         * Get process details from database
         **/
        ResultSet rs = null;
        try {
            String sql = "SELECT process_string, process_variation FROM
apdl.process_maps";
            PreparedStatement psGet = conn.prepareStatement(sql);
            rs = psGet.executeQuery();
       } catch (Exception e) {
            out.println("<h3>" + e + "</h3>");
            out.println("<h3>Failed to retrieve process maps from
database!</h3>");
       }

        /**
         * Store existing service information in an ArrayList
         **/
        ArrayList<String> maps = new ArrayList<String>();
        ArrayList<String> variations = new ArrayList<String>();
        try {
            while (rs.next()) {
                String getMap = rs.getString("process_string");
                maps.add(getMap);
                String getVar = rs.getString("process_variation");
                variations.add(getVar);
            }
            rs.close();
       } catch (Exception e) {
            out.println("<h3>" + e + "</h3>");
            out.println("<h3>Failed to store existing process
information!</h3>");
       }

        /**
         * Find process with smallest variation
         */
        int minIndex = 0;
        double minValue = Double.parseDouble(variations.get(0));
        for (int i=1; i < maps.size(); i++) {
            int tempIndex = i;
            double tempValue = Double.parseDouble(variations.get(i));
            if (tempValue < minValue) {
                minIndex = tempIndex;
                minValue = tempValue;
            }
        }

        /**
         * Execute process
         */
        String process = maps.get(minIndex);
        Pattern pStep = Pattern.compile("[;\\s]+");
        String[] steps = pStep.split(process);
        String input = "0.0";
        String output = null;
        for (int i=0; i<steps.length; i++) {
            Pattern pDetail = Pattern.compile(":");
            String[] details = pDetail.split(steps[i]);

            /**
             * Get web service wsdl locations from registry
             **/
            ResultSet rsWSDL = null;
            try {
                String sql = "SELECT wsdl_location FROM apdl.web_services WHERE
service_name = ?";
                PreparedStatement psGet = conn.prepareStatement(sql);
                psGet.setString(1, details[0]);
                rsWSDL = psGet.executeQuery();
            } catch (Exception e) {
                out.println("<h3>" + e + "</h3>");
                out.println("<h3>Failed to retrieve wsdl location from
registry!</h3>");
            }

            /**
             * Store wsdl location
             **/
            String wsdl = null;
            try {
                while (rsWSDL.next()) {
                    wsdl = rsWSDL.getString("wsdl_location");
                }
                rsWSDL.close();
            } catch (Exception e) {
                out.println("<h3>" + e + "</h3>");
                out.println("<h3>Failed to wsdl location!</h3>");
            }

            try {
                Client client = new Client(new URL(wsdl));
                Object[] servResults = client.invoke(details[1], new Object[]
{details[2], input});
                output = (String)servResults[0];
            } catch (Exception e) {
              out.println("<h3>" + e + "</h3>");
                out.println("<h3>Failed to access web service method!</h3>");
            }
            input = output;
            out.println("<h2>Step #" + (i+1) + " -</h2>");
            out.println("<h3>Service Name: " + details[0] + "</h3>");
            out.println("<h3>Operation Name: " + details[1]  + "</h3>");
            out.println("<h3>Input Type: " + details[2] + "</h3>");
            out.println("<i>Result = " + output + "</i>");
        }

        /**
         * Close database connection
         */
        try {
            conn.close();
        } catch (Exception e) {
            out.println("<h3>" + e + "</h3>");
            out.println("<h3>Failed to close database connection!</h3>");
        }
        out.println("</body>");
        out.println("</html>");
        out.close();
    }

    // <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click
on the + sign on the left to edit the code.">
    /** Handles the HTTP <code>GET</code> method.
     * @param request servlet request
     * @param response servlet response
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

    /** Handles the HTTP <code>POST</code> method.
     * @param request servlet request
     * @param response servlet response
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse
response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

    /** Returns a short description of the servlet.
     */
    public String getServletInfo() {
        return "Short description";
    }
    // </editor-fold>
}
```

ServiceASEI.java

```
package org.apdl.services;

/**
 * This is the service endpoint interface for the ServiceA web service.
 * Created Jun 4, 2006 7:43:53 PM
 * @author APDL
 */

public interface ServiceASEI extends java.rmi.Remote {
    /**
     * Web service operation
     */
    public java.lang.String getServiceName() throws java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getServiceDescription() throws
java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getDatabaseName() throws java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getOperationName() throws java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getInputString() throws java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getOutputString() throws java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getVariationString() throws
java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String addA(String inputType, String inputValue) throws
java.rmi.RemoteException;

}
```

ServiceAImpl.java

```
package org.apdl.services;

import java.sql.*;
import java.util.Random;
import java.util.regex.Pattern;

/**
 * This is the implementation bean class for the ServiceA web service.
 * Created Jun 4, 2006 7:43:53 PM
 * @author APDL
 */
public class ServiceAImpl implements ServiceASEI {
    /**
     * Default constructor for xfire
     */
       public ServiceAImpl() {};

       /**
     * Web service description parameters
     */
    private String serviceName = "ServiceA";
    private String serviceDescription = "This service \"adds A\" to the input
value.";
    private String databaseName = "apdl.service_a";
    private String operationName = "addA";
    private String inputString = "nil, B, D, CD, BC, BCD";
    private String outputString = "A, AB, DA, CDA, ABC, ABCD";
    private String variationString = "0.01, 0.1, 0.1, 0.2, 0.2, 0.3";

    /**
     * Database connection parameters
     */
    private String dbUrl = System.getProperty("APDLdb.url",
"jdbc:derby://localhost:1527/APDLdb");
    private String dbDriver = System.getProperty("APDLdb.driver",
"org.apache.derby.jdbc.ClientDriver");
    private String dbUser = System.getProperty("APDLdb.user", "apdl");
    private String dbPassword = System.getProperty("APDLdb.password", "apdl");

    /**
     * Web service operation
     */
    public java.lang.String getServiceName() throws java.rmi.RemoteException {
        return serviceName;
    }

    /**
     * Web service operation
     */
    public java.lang.String getServiceDescription() throws
java.rmi.RemoteException {
        return serviceDescription;
    }

    /**
     * Web service operation
     */
    public java.lang.String getDatabaseName() throws java.rmi.RemoteException {
        return databaseName;
    }

    /**
     * Web service operation
     */
    public java.lang.String getOperationName() throws java.rmi.RemoteException
{
        return operationName;
    }

    /**
     * Web service operation
     */
    public java.lang.String getInputString() throws java.rmi.RemoteException {
        return inputString;
    }

    /**
     * Web service operation
     */
    public java.lang.String getOutputString() throws java.rmi.RemoteException {
        return outputString;
    }

    /**
     * Web service operation
     */
    public java.lang.String getVariationString() throws
java.rmi.RemoteException {
        return variationString;
    }

    /**
     * Web service operation
     */
    public java.lang.String addA(String inputType, String inputValue) throws
java.rmi.RemoteException {
        double a = 0.0;
        double outputVal = 0.0;
        double var = 0.0;

        double inputVal = Double.parseDouble(inputValue);
        Pattern p = Pattern.compile("[,\\s]+");
        String[] inArray = p.split(inputString);
        String[] varArray = p.split(variationString);
        for (int i=0; i<inArray.length; i++) {
            if (inArray[i].equals(inputType)) {
                var = Double.parseDouble(varArray[i]);
            }
        }

        Random r = new Random();

        a = 1.0 + (r.nextDouble() * 0.01) * Math.pow(-1.0,
(double)r.nextInt(2));

        if (inputVal == 0.0)
            outputVal = a;
        else
            outputVal = (a + inputVal) + (r.nextDouble() * var) * Math.pow(-
1.0, (double)r.nextInt(2));
```

```
        /**
         * Connect to derby database
         **/
        Connection conn = null;
        try {
            Class.forName(dbDriver).newInstance();
            conn = DriverManager.getConnection(dbUrl, dbUser, dbPassword);
       } catch (Exception e) {
            e.printStackTrace();
            System.out.println("Failed to load database driver!");
            return null;
       }

        /**
         * Add transaction to database
         **/
        try {
            String sql = "INSERT INTO apdl.service_a (input_type, input_value,
output_value) VALUES (?, ?, ?)";
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setString(1, inputType);
            ps.setDouble(2, inputVal);
            ps.setDouble(3, outputVal);
            ps.execute();
            conn.close();
       } catch (Exception e) {
            e.printStackTrace();
            System.out.println("Failed to load transaction into database!");
            return null;
       }

        String outputValue = Double.toString(outputVal);
        return outputValue;
    }

}
```

ServiceBSEI.java

```
package org.apdl.services;

/**
 * This is the service endpoint interface for the ServiceB web service.
 * Created Jun 4, 2006 9:34:39 PM
 * @author APDL
 */

public interface ServiceBSEI extends java.rmi.Remote {
    /**
     * Web service operation
     */
    public java.lang.String getServiceName() throws java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getServiceDescription() throws
java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getDatabaseName() throws java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getOperationName() throws java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getInputString() throws java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getOutputString() throws java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getVariationString() throws
java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String addB(String inputType, String inputValue) throws
java.rmi.RemoteException;

}
```

ServiceBImpl.java

```
package org.apdl.services;

import java.sql.*;
import java.util.Random;
import java.util.regex.Pattern;

/**
 * This is the implementation bean class for the ServiceB web service.
 * Created Jun 4, 2006 9:34:39 PM
 * @author APDL
 */
public class ServiceBImpl implements ServiceBSEI {
    /**
     * Default constructor for xfire
     */
      public ServiceBImpl() {};

      /**
     * Web service description parameters
     */
    private String serviceName = "ServiceB";
    private String serviceDescription = "This service \"adds B\" to the input
value.";
    private String databaseName = "apdl.service_b";
    private String operationName = "addB";
    private String inputString = "nil, A, C, DA, CD, CDA";
    private String outputString = "B, AB, BC, DAB, BCD, ABCD";
    private String variationString = "0.01, 0.15, 0.15, 0.15, 0.15, 0.2";

    /**
     * Database connection parameters
     */
    private String dbUrl = System.getProperty("APDLdb.url",
"jdbc:derby://localhost:1527/APDLdb");
    private String dbDriver = System.getProperty("APDLdb.driver",
"org.apache.derby.jdbc.ClientDriver");
    private String dbUser = System.getProperty("APDLdb.user", "apdl");
    private String dbPassword = System.getProperty("APDLdb.password", "apdl");

    /**
     * Web service operation
     */
    public java.lang.String getServiceName() throws java.rmi.RemoteException {
        return serviceName;
    }

    /**
     * Web service operation
     */
    public java.lang.String getServiceDescription() throws
java.rmi.RemoteException {
        return serviceDescription;
    }

    /**
     * Web service operation
     */
    public java.lang.String getDatabaseName() throws java.rmi.RemoteException {
        return databaseName;
    }

    /**
     * Web service operation
     */
    public java.lang.String getOperationName() throws java.rmi.RemoteException
{
        return operationName;
    }

    /**
     * Web service operation
     */
    public java.lang.String getInputString() throws java.rmi.RemoteException {
        return inputString;
    }

    /**
     * Web service operation
     */
    public java.lang.String getOutputString() throws java.rmi.RemoteException {
        return outputString;
    }

    /**
     * Web service operation
     */
    public java.lang.String getVariationString() throws
java.rmi.RemoteException {
        return variationString;
    }

    /**
     * Web service operation
     */
    public java.lang.String addB(String inputType, String inputValue) throws
java.rmi.RemoteException {
        double b = 0.0;
        double outputVal = 0.0;
        double var = 0.0;

        double inputVal = Double.parseDouble(inputValue);
        Pattern p = Pattern.compile("[,\\s]+");
        String[] inArray = p.split(inputString);
        String[] varArray = p.split(variationString);
        for (int i=0; i<inArray.length; i++) {
            if (inArray[i].equals(inputType)) {
                var = Double.parseDouble(varArray[i]);
            }
        }

        Random r = new Random();

        b = 2.0 + (r.nextDouble() * 0.01) * Math.pow(-1.0,
(double)r.nextInt(2));

        if (inputVal == 0.0)
            outputVal = b;
        else
            outputVal = (b + inputVal) + (r.nextDouble() * var) * Math.pow(-
1.0, (double)r.nextInt(2));
```

```
        /**
         * Connect to derby database
         **/
        Connection conn = null;
        try {
            Class.forName(dbDriver).newInstance();
            conn = DriverManager.getConnection(dbUrl, dbUser, dbPassword);
       } catch (Exception e) {
            e.printStackTrace();
            System.out.println("Failed to load database driver!");
            return null;
       }

        /**
         * Add transaction to database
         **/
        try {
            String sql = "INSERT INTO apdl.service_b (input_type, input_value,
output_value) VALUES (?, ?, ?)";
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setString(1, inputType);
            ps.setDouble(2, inputVal);
            ps.setDouble(3, outputVal);
            ps.execute();
            conn.close();
       } catch (Exception e) {
            e.printStackTrace();
            System.out.println("Failed to load transaction into database!");
            return null;
       }

        String outputValue = Double.toString(outputVal);
        return outputValue;
    }

}
```

ServiceCSEI.java

```
package org.apdl.services;

/**
 * This is the service endpoint interface for the ServiceC web service.
 * Created Jun 4, 2006 9:35:06 PM
 * @author APDL
 */

public interface ServiceCSEI extends java.rmi.Remote {
    /**
     * Web service operation
     */
    public java.lang.String getServiceName() throws java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getServiceDescription() throws
java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getDatabaseName() throws java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getOperationName() throws java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getInputString() throws java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getOutputString() throws java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getVariationString() throws
java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String addC(String inputType, String inputValue) throws
java.rmi.RemoteException;

}
```

ServiceCImpl.java

```
package org.apdl.services;

import java.sql.*;
import java.util.Random;
import java.util.regex.Pattern;

/**
 * This is the implementation bean class for the ServiceC web service.
 * Created Jun 4, 2006 9:35:06 PM
 * @author APDL
 */
public class ServiceCImpl implements ServiceCSEI {
    /**
     * Default constructor for xfire
     */
      public ServiceCImpl() {};

      /**
     * Web service description parameters
     */
    private String serviceName = "ServiceC";
    private String serviceDescription = "This service \"adds C\" to the input
value.";
    private String databaseName = "apdl.service_c";
    private String operationName = "addC";
    private String inputString = "nil, B, D, AB, DA, DAB";
    private String outputString = "C, BC, CD, ABC, CDA, ABCD";
    private String variationString = "0.01, 0.05, 0.05, 0.1, 0.1, 0.1";

    /**
     * Database connection parameters
     */
    private String dbUrl = System.getProperty("APDLdb.url",
"jdbc:derby://localhost:1527/APDLdb");
    private String dbDriver = System.getProperty("APDLdb.driver",
"org.apache.derby.jdbc.ClientDriver");
    private String dbUser = System.getProperty("APDLdb.user", "apdl");
    private String dbPassword = System.getProperty("APDLdb.password", "apdl");

    /**
     * Web service operation
     */
    public java.lang.String getServiceName() throws java.rmi.RemoteException {
        return serviceName;
    }

    /**
     * Web service operation
     */
    public java.lang.String getServiceDescription() throws
java.rmi.RemoteException {
        return serviceDescription;
    }

    /**
     * Web service operation
     */
    public java.lang.String getDatabaseName() throws java.rmi.RemoteException {
        return databaseName;
    }

    /**
     * Web service operation
     */
    public java.lang.String getOperationName() throws java.rmi.RemoteException
{
        return operationName;
    }

    /**
     * Web service operation
     */
    public java.lang.String getInputString() throws java.rmi.RemoteException {
        return inputString;
    }

    /**
     * Web service operation
     */
    public java.lang.String getOutputString() throws java.rmi.RemoteException {
        return outputString;
    }

    /**
     * Web service operation
     */
    public java.lang.String getVariationString() throws
java.rmi.RemoteException {
        return variationString;
    }

    /**
     * Web service operation
     */
    public java.lang.String addC(String inputType, String inputValue) throws
java.rmi.RemoteException {
        double c = 0.0;
        double outputVal = 0.0;
        double var = 0.0;

        double inputVal = Double.parseDouble(inputValue);
        Pattern p = Pattern.compile("[,\\s]+");
        String[] inArray = p.split(inputString);
        String[] varArray = p.split(variationString);
        for (int i=0; i<inArray.length; i++) {
            if (inArray[i].equals(inputType)) {
                var = Double.parseDouble(varArray[i]);
            }
        }

        Random r = new Random();

        c = 3.0 + (r.nextDouble() * 0.01) * Math.pow(-1.0,
(double)r.nextInt(2));

        if (inputVal == 0.0)
            outputVal = c;
        else
            outputVal = (c + inputVal) + (r.nextDouble() * var) * Math.pow(-
1.0, (double)r.nextInt(2));
```

```
        /**
         * Connect to derby database
         **/
        Connection conn = null;
        try {
            Class.forName(dbDriver).newInstance();
            conn = DriverManager.getConnection(dbUrl, dbUser, dbPassword);
       } catch (Exception e) {
            e.printStackTrace();
            System.out.println("Failed to load database driver!");
            return null;
       }

        /**
         * Add transaction to database
         **/
        try {
            String sql = "INSERT INTO apdl.service_c (input_type, input_value,
output_value) VALUES (?, ?, ?)";
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setString(1, inputType);
            ps.setDouble(2, inputVal);
            ps.setDouble(3, outputVal);
            ps.execute();
            conn.close();
       } catch (Exception e) {
            e.printStackTrace();
            System.out.println("Failed to load transaction into database!");
            return null;
       }

        String outputValue = Double.toString(outputVal);
        return outputValue;
    }

}
```

ServiceDSEI.java

```
package org.apdl.services;

/**
 * This is the service endpoint interface for the ServiceD web service.
 * Created Jun 4, 2006 9:35:23 PM
 * @author APDL
 */

public interface ServiceDSEI extends java.rmi.Remote {
    /**
     * Web service operation
     */
    public java.lang.String getServiceName() throws java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getServiceDescription() throws
java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getDatabaseName() throws java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getOperationName() throws java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getInputString() throws java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getOutputString() throws java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getVariationString() throws
java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String addD(String inputType, String inputValue) throws
java.rmi.RemoteException;
}
```

ServiceDImpl.java

```
package org.apdl.services;

import java.sql.*;
import java.util.Random;
import java.util.regex.Pattern;

/**
 * This is the implementation bean class for the ServiceD web service.
 * Created Jun 4, 2006 9:35:23 PM
 * @author APDL
 */
public class ServiceDImpl implements ServiceDSEI {
    /**
     * Default constructor for xfire
     */
      public ServiceDImpl() {};

      /**
     * Web service description parameters
     */
    private String serviceName = "ServiceD";
    private String serviceDescription = "This service \"adds D\" to the input
value.";
    private String databaseName = "apdl.service_d";
    private String operationName = "addD";
    private String inputString = "nil, A, C, AB, BC, ABC";
    private String outputString = "D, DA, CD, DAB, BCD, ABCD";
    private String variationString = "0.02, 0.1, 0.1, 0.2, 0.2, 0.2";

    /**
     * Database connection parameters
     */
    private String dbUrl = System.getProperty("APDLdb.url",
"jdbc:derby://localhost:1527/APDLdb");
    private String dbDriver = System.getProperty("APDLdb.driver",
"org.apache.derby.jdbc.ClientDriver");
    private String dbUser = System.getProperty("APDLdb.user", "apdl");
    private String dbPassword = System.getProperty("APDLdb.password", "apdl");

    /**
     * Web service operation
     */
    public java.lang.String getServiceName() throws java.rmi.RemoteException {
        return serviceName;
    }

    /**
     * Web service operation
     */
    public java.lang.String getServiceDescription() throws
java.rmi.RemoteException {
        return serviceDescription;
    }

    /**
     * Web service operation
     */
    public java.lang.String getDatabaseName() throws java.rmi.RemoteException {
        return databaseName;
    }

    /**
     * Web service operation
     */
    public java.lang.String getOperationName() throws java.rmi.RemoteException
{
        return operationName;
    }

    /**
     * Web service operation
     */
    public java.lang.String getInputString() throws java.rmi.RemoteException {
        return inputString;
    }

    /**
     * Web service operation
     */
    public java.lang.String getOutputString() throws java.rmi.RemoteException {
        return outputString;
    }

    /**
     * Web service operation
     */
    public java.lang.String getVariationString() throws
java.rmi.RemoteException {
        return variationString;
    }

    /**
     * Web service operation
     */
    public java.lang.String addD(String inputType, String inputValue) throws
java.rmi.RemoteException {
        double d = 0.0;
        double outputVal = 0.0;
        double var = 0.0;

        double inputVal = Double.parseDouble(inputValue);
        Pattern p = Pattern.compile("[,\\s]+");
        String[] inArray = p.split(inputString);
        String[] varArray = p.split(variationString);
        for (int i=0; i<inArray.length; i++) {
            if (inArray[i].equals(inputType)) {
                var = Double.parseDouble(varArray[i]);
            }
        }

        Random r = new Random();

        d = 4.0 + (r.nextDouble() * 0.01) * Math.pow(-1.0,
(double)r.nextInt(2));

        if (inputVal == 0.0)
            outputVal = d;
        else
            outputVal = (d + inputVal) + (r.nextDouble() * var) * Math.pow(-
1.0, (double)r.nextInt(2));
```

```
        /**
         * Connect to derby database
         **/
        Connection conn = null;
        try {
            Class.forName(dbDriver).newInstance();
            conn = DriverManager.getConnection(dbUrl, dbUser, dbPassword);
       } catch (Exception e) {
            e.printStackTrace();
            System.out.println("Failed to load database driver!");
            return null;
       }

        /**
         * Add transaction to database
         **/
        try {
            String sql = "INSERT INTO apdl.service_d (input_type, input_value, 
output_value) VALUES (?, ?, ?)";
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setString(1, inputType);
            ps.setDouble(2, inputVal);
            ps.setDouble(3, outputVal);
            ps.execute();
            conn.close();
       } catch (Exception e) {
            e.printStackTrace();
            System.out.println("Failed to load transaction into database!");
            return null;
       }

        String outputValue = Double.toString(outputVal);
        return outputValue;
    }

}
```

AltServiceBSEI.java

```
package org.apdl.services;

/**
 * This is the service endpoint interface for the ServiceB web service.
 * Created Jun 4, 2006 9:34:39 PM
 * @author APDL
 */

public interface AltServiceBSEI extends java.rmi.Remote {
    /**
     * Web service operation
     */
    public java.lang.String getServiceName() throws java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getServiceDescription() throws
java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getDatabaseName() throws java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getOperationName() throws java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getInputString() throws java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getOutputString() throws java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getVariationString() throws
java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String altAddB(String inputType, String inputValue) throws
java.rmi.RemoteException;

}
```

AltServiceBImpl.java

```
package org.apdl.services;

import java.sql.*;
import java.util.Random;
import java.util.regex.Pattern;

/**
 * This is the implementation bean class for the ServiceB web service.
 * Created Jun 4, 2006 9:34:39 PM
 * @author APDL
 */
public class AltServiceBImpl implements AltServiceBSEI {
    /**
     * Default constructor for xfire
     */
      public AltServiceBImpl() {};

      /**
     * Web service description parameters
     */
    private String serviceName = "AltServiceB";
    private String serviceDescription = "This is an alternate service that
\"adds B\" to the input value.";
    private String databaseName = "apdl.alt_service_b";
    private String operationName = "altAddB";
    private String inputString = "nil, A, C, DA, CD, CDA";
    private String outputString = "B, AB, BC, DAB, BCD, ABCD";
    private String variationString = "0.01, 0.01, 0.01, 0.02, 0.02, 0.03";

    /**
     * Database connection parameters
     */
    private String dbUrl = System.getProperty("APDLdb.url",
"jdbc:derby://localhost:1527/APDLdb");
    private String dbDriver = System.getProperty("APDLdb.driver",
"org.apache.derby.jdbc.ClientDriver");
    private String dbUser = System.getProperty("APDLdb.user", "apdl");
    private String dbPassword = System.getProperty("APDLdb.password", "apdl");

    /**
     * Web service operation
     */
    public java.lang.String getServiceName() throws java.rmi.RemoteException {
        return serviceName;
    }

    /**
     * Web service operation
     */
    public java.lang.String getServiceDescription() throws
java.rmi.RemoteException {
        return serviceDescription;
    }

    /**
     * Web service operation
     */
    public java.lang.String getDatabaseName() throws java.rmi.RemoteException {
        return databaseName;
    }

    /**
     * Web service operation
     */
    public java.lang.String getOperationName() throws java.rmi.RemoteException
{
        return operationName;
    }

    /**
     * Web service operation
     */
    public java.lang.String getInputString() throws java.rmi.RemoteException {
        return inputString;
    }

    /**
     * Web service operation
     */
    public java.lang.String getOutputString() throws java.rmi.RemoteException {
        return outputString;
    }

    /**
     * Web service operation
     */
    public java.lang.String getVariationString() throws
java.rmi.RemoteException {
        return variationString;
    }

    /**
     * Web service operation
     */
    public java.lang.String altAddB(String inputType, String inputValue) throws
java.rmi.RemoteException {
        double b = 0.0;
        double outputVal = 0.0;
        double var = 0.0;

        double inputVal = Double.parseDouble(inputValue);
        Pattern p = Pattern.compile("[,\\s]+");
        String[] inArray = p.split(inputString);
        String[] varArray = p.split(variationString);
        for (int i=0; i<inArray.length; i++) {
            if (inArray[i].equals(inputType)) {
                var = Double.parseDouble(varArray[i]);
            }
        }

        Random r = new Random();

        b = 2.0 + (r.nextDouble() * 0.01) * Math.pow(-1.0,
(double)r.nextInt(2));

        if (inputVal == 0.0)
            outputVal = b;
        else
            outputVal = (b + inputVal) + (r.nextDouble() * var) * Math.pow(-
1.0, (double)r.nextInt(2));
```

```
        /**
         * Connect to derby database
         **/
        Connection conn = null;
        try {
            Class.forName(dbDriver).newInstance();
            conn = DriverManager.getConnection(dbUrl, dbUser, dbPassword);
       } catch (Exception e) {
            e.printStackTrace();
            System.out.println("Failed to load database driver!");
            return null;
       }

        /**
         * Add transaction to database
         **/
        try {
            String sql = "INSERT INTO apdl.alt_service_b (input_type,
input_value, output_value) VALUES (?, ?, ?)";
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setString(1, inputType);
            ps.setDouble(2, inputVal);
            ps.setDouble(3, outputVal);
            ps.execute();
            conn.close();
       } catch (Exception e) {
            e.printStackTrace();
            System.out.println("Failed to load transaction into database!");
            return null;
       }

        String outputValue = Double.toString(outputVal);
        return outputValue;
    }

}
```

APPENDIX C IMPELLER EXAMPLE – CODE

Services.xml

```
<beans xmlns="http://xfire.codehaus.org/config/1.0">

  <service>
    <name>CATIAService</name>
    <namespace>APDLServices-1.0/services</namespace>
    <serviceClass>org.apdl.services.CATIAServiceSEI</serviceClass>

<implementationClass>org.apdl.services.CATIAServiceImpl</implementationClass>
  </service>

  <service>
    <name>HyperMeshService</name>
    <namespace>APDLServices-1.0/services</namespace>
    <serviceClass>org.apdl.services.HyperMeshServiceSEI</serviceClass>

<implementationClass>org.apdl.services.HyperMeshServiceImpl</implementationClas
s>
  </service>

  <service>
    <name>FluentService</name>
    <namespace>APDLServices-1.0/services</namespace>
    <serviceClass>org.apdl.services.FluentServiceSEI</serviceClass>

<implementationClass>org.apdl.services.FluentServiceImpl</implementationClass>
  </service>

  <service>
    <name>ANSYSService</name>
    <namespace>APDLServices-1.0/services</namespace>
    <serviceClass>org.apdl.services.ANSYSServiceSEI</serviceClass>

<implementationClass>org.apdl.services.ANSYSServiceImpl</implementationClass>
  </service>

</beans>
```

WSRegistry.java

```
/*
 * WSRegistry.java
 *
 * Created on June 6, 2006, 7:26 PM
 */

package org.apdl.servlets;

import java.io.*;

import javax.servlet.*;
import javax.servlet.http.*;

import java.sql.*;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.regex.*;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;

/**
 *
 * @author APDL
 * @version
 */
public class WSRegistry extends HttpServlet {
    /**
     * Parameters for "services.xml" file search
     */
    private PrintWriter out;
    private static String webServicesFile = "services.xml";
    private static String filePath = "C:/dev/projects/APDLServices";
    private boolean fileFound = false;
    private String webServiceDoc = null;

    /** Processes requests for both HTTP <code>GET</code> and <code>POST</code>
methods.
     * @param request servlet request
     * @param response servlet response
     */
    protected void processRequest(HttpServletRequest request,
HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet WSRegistry</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Servlet WSRegistry at " + request.getContextPath () +
"</h1>");

        /**
         * Database connection parameters
         */
        String dbUrl = System.getProperty("APDLdb.url",
"jdbc:derby://localhost:1527/APDLdb");
        String dbDriver = System.getProperty("APDLdb.driver",
"org.apache.derby.jdbc.ClientDriver");
        String dbUser = System.getProperty("APDLdb.user", "apdl");
        String dbPassword = System.getProperty("APDLdb.password", "apdl");

        /**
         * Connect to derby database
         **/
        Connection conn = null;
        try {
            Class.forName(dbDriver).newInstance();
            conn = DriverManager.getConnection(dbUrl, dbUser, dbPassword);
       } catch (Exception e) {
            out.println("<h3>" + e + "</h3>");
            out.println("<h3>Failed to load database driver!</h3>");
       }

        /**
         * Get existing services from database
         **/
        ResultSet rs = null;
        try {
            String sql = "SELECT service_name FROM apdl.web_services";
            PreparedStatement psGet = conn.prepareStatement(sql);
            rs = psGet.executeQuery();
       } catch (Exception e) {
            out.println("<h3>" + e + "</h3>");
            out.println("<h3>Failed to retrieve services from database!</h3>");
       }

        /**
         * Store existing service information in an ArrayList
         **/
        ArrayList<String> existingServices = new ArrayList<String>();
        boolean databaseEmpty = true;
        try {
            while (rs.next()) {
                String service = rs.getString("service_name");
                existingServices.add(service);
                databaseEmpty = false;
            }
            rs.close();
       } catch (Exception e) {
            out.println("<h3>" + e + "</h3>");
            out.println("<h3>Failed to store existing service
information!</h3>");
       }

        /**
         * Find "services.xml" deployment description file
         */
        File dir = new File(filePath);
        try {
            findFile(dir);
        } catch (Exception e) {
            out.println("<h3>" + e + "</h3>");
            out.println("<h3>Failed to find file \"services.xml\"!</h3>");
        }

        Document webServices = null;
        try {
            File servicesXML = new File(webServiceDoc);
            DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
            dbf.setValidating(false);
            dbf.setNamespaceAware(true);
            DocumentBuilder documentBuilder = dbf.newDocumentBuilder();
            webServices = documentBuilder.parse(servicesXML);
        } catch (Exception e) {
            out.println("<h3>" + e + "</h3>");
            out.println("<h3>Failed to parse file \"services.xml\"!</h3>");
        }

        /**
         * Parse web service names
         */
        ArrayList<String> serviceNames = new ArrayList<String>();
        ArrayList<String> serviceNamespaces = new ArrayList<String>();
        NodeList nl = webServices.getElementsByTagName("service");
        for (int n = 0; n < nl.getLength(); n++) {
            Element service = (Element) nl.item(n);
            String name =
service.getElementsByTagName("name").item(0).getFirstChild().getNodeValue();
            String namespace =
service.getElementsByTagName("namespace").item(0).getFirstChild().getNodeValue(
);

            if (databaseEmpty == false) {
                boolean matchFound = false;
                Iterator iter = existingServices.iterator();
                while (matchFound != true && iter.hasNext()) {
                    String existingName = (String)iter.next();
                    if (name.equals(existingName))
                        matchFound = true;
                }
                if (matchFound == false) {
                    serviceNames.add(name);
                    serviceNamespaces.add(namespace);
                }
            }

            if (databaseEmpty == true) {
                serviceNames.add(name);
                serviceNamespaces.add(namespace);
            }
        }

        /**
         * Get url for servlet context
         */
        StringBuffer url = new StringBuffer();
        url.append(request.getScheme() + "://");
        url.append(request.getServerName());
        if(request.getServerPort()!=80)
            url.append(":" + request.getServerPort());
        String uri = url.toString();

        /**
         * Add services to database
         **/
        for (int i=0; i<serviceNames.size(); i++) {
            String serviceName = serviceNames.get(i);
            String serviceNamespace = serviceNamespaces.get(i);
            String wsdlLocation = uri + "/" + serviceNamespace + "/" +
serviceName + "?wsdl";

            try {
                String sql = "INSERT INTO apdl.web_services (service_name,
wsdl_location) VALUES (?, ?)";
                PreparedStatement psSet = conn.prepareStatement(sql);
                psSet.setString(1, serviceName);
                psSet.setString(2, wsdlLocation);
                psSet.execute();
            } catch (Exception e) {
                out.println("<h3>" + e + "</h3>");
                out.println("<h3>Failed to load service(s) into
database!</h3>");
            }
            out.print("<h3><i>" + serviceName + "</i> added to the
registry.</h3>");
        }

        /**
         * Close database connection
         */
        try {
            conn.close();
        } catch (Exception e) {
            out.println("<h3>" + e + "</h3>");
            out.println("<h3>Error during file search!</h3>");
        }

        /**
         * Close HTML output
         */
        out.println("</body>");
        out.println("</html>");
        out.close();
    }

    // <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click
on the + sign on the left to edit the code.">
    /** Handles the HTTP <code>GET</code> method.
     * @param request servlet request
     * @param response servlet response
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

    /** Handles the HTTP <code>POST</code> method.
     * @param request servlet request
     * @param response servlet response
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse
response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

    /** Returns a short description of the servlet.
     */
    public String getServletInfo() {
        return "Short description";
    }
    // </editor-fold>

    /**
     * Search method for finding the file "services.xml"
     */
    protected void findFile(File dir) {
        if (fileFound == false) {
            try {
                if (dir.isDirectory()) {
                    String[] children = dir.list();
                    String filename;
                    if (children != null) {
                        for (int i=0; i<children.length; i++) {
                            filename = children[i];
                            File dirChild = new File(dir, children[i] + "\\");
                            if (dirChild.isDirectory()) {
                                findFile(dirChild);
                            } else {
                                if (filename.equals(webServicesFile)) {
                                    webServiceDoc = dir + "\\" + filename;
                                    fileFound = true;
                                }
                            }
                        }
                    }
                }
            } catch (Exception e) {
                out.println("<h3>" + e + "</h3>");
                out.println("<h3>Failed to close database connection!</h3>");
            }
        }
    }

}
```

KnowledgeAgent.java

```
/*
 * KnowledgeAgent.java
 *
 * Created on June 6, 2006, 11:29 PM
 */

package org.apdl.agents;

import java.io.*;
import java.net.URL;

import javax.servlet.*;
import javax.servlet.http.*;

import java.sql.*;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.regex.Pattern;
import org.codehaus.xfire.client.Client;

/**
 *
 * @author APDL
 * @version
 */
public class KnowledgeAgent extends HttpServlet {

    /** Processes requests for both HTTP <code>GET</code> and <code>POST</code>
methods.
     * @param request servlet request
     * @param response servlet response
     */
    protected void processRequest(HttpServletRequest request,
HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet KnowledgeAgent</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Servlet KnowledgeAgent at " + request.getContextPath
() + "</h1>");

        /**
         * Database connection parameters
         */
        String dbUrl = System.getProperty("APDLdb.url",
"jdbc:derby://localhost:1527/APDLdb");
        String dbDriver = System.getProperty("APDLdb.driver",
"org.apache.derby.jdbc.ClientDriver");
        String dbUser = System.getProperty("APDLdb.user", "apdl");
        String dbPassword = System.getProperty("APDLdb.password", "apdl");

        /**
         * Connect to derby database
         **/
        Connection conn = null;
        try {
            Class.forName(dbDriver).newInstance();
            conn = DriverManager.getConnection(dbUrl, dbUser, dbPassword);
       } catch (Exception e) {
            out.println("<h3>" + e + "</h3>");
            out.println("<h3>Failed to load database driver!</h3>");
       }

        /**
         * Get web service wsdl locations from registry
         **/
        ResultSet rs = null;
        try {
            String sql = "SELECT wsdl_location FROM apdl.web_services";
            PreparedStatement psGet = conn.prepareStatement(sql);
            rs = psGet.executeQuery();
       } catch (Exception e) {
            out.println("<h3>" + e + "</h3>");
            out.println("<h3>Failed to retrieve services from database!</h3>");
       }

        /**
         * Store existing service information in an ArrayList
         **/
        ArrayList<String> serviceInfo = new ArrayList<String>();
        try {
            while (rs.next()) {
                String wsdl = rs.getString("wsdl_location");
                serviceInfo.add(wsdl);
            }
            rs.close();
       } catch (Exception e) {
            out.println("<h3>" + e + "</h3>");
            out.println("<h3>Failed to store existing service
information!</h3>");
       }

        for (Iterator iter = serviceInfo.iterator(); iter.hasNext();) {
            String wsdlLocation = (String)iter.next();
            String serviceName = null;
            String serviceDescription = null;
            String operationName = null;
            String inputString = null;
            String outputString = null;

            try {
                Client client = new Client(new URL(wsdlLocation));
                Object[] servResults = client.invoke("getServiceName", new
Object[] {});
                serviceName = (String)servResults[0];

                Object[] descResults = client.invoke("getServiceDescription",
new Object[] {});
                serviceDescription = (String)descResults[0];

                Object[] operResults = client.invoke("getOperationName", new
Object[] {});
                operationName = (String)operResults[0];

                Object[] inResults = client.invoke("getInputString", new
Object[] {});
                inputString = (String)inResults[0];
                Object[] outResults = client.invoke("getOutputString", new
Object[] {});
                outputString = (String)outResults[0];

            } catch (Exception e) {
              out.println("<h3>" + e + "</h3>");
                out.println("<h3>Failed to access web service reporting
methods!</h3>");
            }

            try {
                String sql = "UPDATE apdl.web_services SET service_name=?,
service_description=?, " +
                             "operation_name=?, input_string=?, output_string=?
WHERE wsdl_location=?";
                PreparedStatement psSet = conn.prepareStatement(sql);
                psSet.setString(1, serviceName);
                psSet.setString(2, serviceDescription);
                psSet.setString(3, operationName);
                psSet.setString(4, inputString);
                psSet.setString(5, outputString);
                psSet.setString(6, wsdlLocation);
                psSet.execute();
            } catch (Exception e) {
                out.println("<h3>" + e + "</h3>");
                out.println("<h3>Failed to update web service(s)!</h3>");
            }
            out.print("<h3>Registry entry for <i>" + serviceName + "</i>
updated.</h3>");
        }

        /**
         * Close database connection
         */
        try {
            conn.close();
        } catch (Exception e) {
            out.println("<h3>" + e + "</h3>");
            out.println("<h3>Failed to close database connection!</h3>");
        }

        out.println("</body>");
        out.println("</html>");
        out.close();
    }

    // <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click
on the + sign on the left to edit the code.">
    /** Handles the HTTP <code>GET</code> method.
     * @param request servlet request
     * @param response servlet response
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

    /** Handles the HTTP <code>POST</code> method.
     * @param request servlet request
     * @param response servlet response
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse
response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

    /** Returns a short description of the servlet.
     */
    public String getServletInfo() {
        return "Short description";
    }
    // </editor-fold>
}
```

ConfigurationAgent.java

```
/*
 * ConfigurationAgent.java
 *
 * Created on June 7, 2006, 12:09 AM
 */

package org.apdl.agents;

import java.io.*;
import java.net.URL;

import javax.servlet.*;
import javax.servlet.http.*;

import java.sql.*;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.regex.Pattern;
import org.codehaus.xfire.client.Client;

/**
 *
 * @author APDL
 * @version
 */
public class ConfigurationAgent extends HttpServlet {

    private Connection conn = null;
    private PrintWriter out;

    private ArrayList<String> servNames;
    private ArrayList<String> operNames;
    private ArrayList<String> inStrings;
    private ArrayList<String> outStrings;
    private ArrayList<String> existingMaps;

    /** Processes requests for both HTTP <code>GET</code> and <code>POST</code>
methods.
     * @param request servlet request
     * @param response servlet response
     */
    protected void processRequest(HttpServletRequest request,
HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        //PrintWriter out = response.getWriter();
        out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet ConfigurationAgent</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Servlet ConfigurationAgent at " +
request.getContextPath () + "</h1>");

        /**
         * Database connection parameters
         */
        String dbUrl = System.getProperty("APDLdb.url",
"jdbc:derby://localhost:1527/APDLdb");
        String dbDriver = System.getProperty("APDLdb.driver",
"org.apache.derby.jdbc.ClientDriver");
        String dbUser = System.getProperty("APDLdb.user", "apdl");
        String dbPassword = System.getProperty("APDLdb.password", "apdl");

        /**
         * Connect to derby database
         **/
        try {
            Class.forName(dbDriver).newInstance();
            conn = DriverManager.getConnection(dbUrl, dbUser, dbPassword);
       } catch (Exception e) {
            out.println("<h3>" + e + "</h3>");
            out.println("<h3>Failed to load database driver!</h3>");
       }

        /**
         * Get web service information from registry
         **/
        ResultSet rsServ = null;
        try {
            String sql = "SELECT service_name, operation_name, input_string,
output_string FROM apdl.web_services";
            PreparedStatement psGet = conn.prepareStatement(sql);
            rsServ = psGet.executeQuery();
       } catch (Exception e) {
            out.println("<h3>" + e + "</h3>");
            out.println("<h3>Failed to retrieve services from database!</h3>");
       }

        /**
         * Store existing service information in an ArrayList
         **/
        servNames = new ArrayList<String>();
        operNames = new ArrayList<String>();
        inStrings = new ArrayList<String>();
        outStrings = new ArrayList<String>();
        try {
            while (rsServ.next()) {
                String serv = rsServ.getString("service_name");
                servNames.add(serv);

                /* Check for multiple service operaitions */
                String oper = rsServ.getString("operation_name");
                if (oper.indexOf(';') != -1) {
                    String input = rsServ.getString("input_string");
                    String output = rsServ.getString("output_string");
                    Pattern p = Pattern.compile(";");
                    String[] operArray = p.split(oper);
                    String[] inArray = p.split(input);
                    String[] outArray = p.split(output);

                    operNames.add(operArray[0]);
                    inStrings.add(inArray[0]);
                    outStrings.add(outArray[0]);
                    for (int i=1; i<operArray.length; i++) {
                        servNames.add(serv);
                        operNames.add(operArray[i]);
                        inStrings.add(inArray[i]);
                        outStrings.add(outArray[i]);
                    }
                } else {
                    operNames.add(oper);
                    String input = rsServ.getString("input_string");
                    inStrings.add(input);
                    String output = rsServ.getString("output_string");
                    outStrings.add(output);
                }
            }
            rsServ.close();
       } catch (Exception e) {
            out.println("<h3>" + e + "</h3>");
            out.println("<h3>Failed to store existing service
information!</h3>");
       }

        /**
         * Get existing process maps from the database
         */
        ResultSet rsMaps = null;
        try {
            String sql = "SELECT process_string FROM apdl.process_maps";
            PreparedStatement psGet = conn.prepareStatement(sql);
            rsMaps = psGet.executeQuery();
       } catch (Exception e) {
            out.println("<h3>" + e + "</h3>");
            out.println("<h3>Failed to retrieve process maps from
database!</h3>");
       }

        /**
         * Store existing process information in an ArrayList
         **/
        existingMaps = new ArrayList<String>();
        try {
            while (rsMaps.next()) {
                String map = rsMaps.getString("process_string");
                existingMaps.add(map);
            }
            rsMaps.close();
       } catch (Exception e) {
            out.println("<h3>" + e + "</h3>");
            out.println("<h3>Failed to store existing process
information!</h3>");
       }

        /**
         * Map and store process paths
         */
        String endpoint = request.getParameter("endpoint");
        mapProcess("", endpoint);

        /**
         * Close database connection
         */
        try {
            conn.close();
        } catch (Exception e) {
            out.println("<h3>" + e + "</h3>");
            out.println("<h3>Failed to close database connection!</h3>");
        }

        out.println("</body>");
        out.println("</html>");
        out.close();
    }

    // <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click
on the + sign on the left to edit the code.">
    /** Handles the HTTP <code>GET</code> method.
     * @param request servlet request
     * @param response servlet response
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

    /** Handles the HTTP <code>POST</code> method.
     * @param request servlet request
     * @param response servlet response
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse
response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

    /** Returns a short description of the servlet.
     */
    public String getServletInfo() {
        return "Short description";
    }
    // </editor-fold>

    /**
     * Method for mapping possible process paths and storing results in a
database
     */
    protected void mapProcess(String mapString, String findNext) {
        if (findNext.equals("nil")) {
            boolean matchFound = false;
            Iterator iter = existingMaps.iterator();
            while (matchFound != true && iter.hasNext()) {
                String existingMap = (String)iter.next();
                if (mapString.equals(existingMap))
                    matchFound = true;
            }
            if (matchFound == false) {
                try {
                    String sql = "INSERT INTO apdl.process_maps " +
                            "(process_string) VALUES (?)";
                    PreparedStatement psSet = conn.prepareStatement(sql);
                    psSet.setString(1, mapString);
                    psSet.execute();
                } catch (Exception e) {
                    out.println("<h3>" + e + "</h3>");
                    out.println("<h3>Failed to load new process into
database!</h3>");
                }

                out.println("<h2>Process added to database!</h2>");
                Pattern pStep = Pattern.compile("[;\\s]+");
                String[] steps = pStep.split(mapString);
                for (int i=0; i<steps.length; i++) {
                    out.println("<h3>Step #" + (i+1) + " - " + steps[i] +
"</h3>");
                }
                out.println("</br>");
            }
            return;
        }

        for (int i=0; i<operNames.size(); i++) {
            String name = servNames.get(i);
            String operation = operNames.get(i);
            String tempIn = inStrings.get(i);
            String tempOut = outStrings.get(i);

            Pattern p = Pattern.compile("[,\\s]+");
            String[] inputs = p.split(tempIn);
            String[] outputs = p.split(tempOut);

            for (int j=0; j<outputs.length; j++) {
                if (outputs[j].equals(findNext)) {
                    String newMap = name + ":" + operation + ":" + inputs[j] +
"; " + mapString;
                    mapProcess(newMap, inputs[j]);
                }
            }
        }
    }

}
```

ExecutionAgent.java

```
/*
 * ExecutionAgent.java
 *
 * Created on June 7, 2006, 4:35 PM
 */

package org.apdl.agents;

import java.io.*;
import java.net.*;

import javax.servlet.*;
import javax.servlet.http.*;

import java.sql.*;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.regex.Pattern;
import org.codehaus.xfire.client.Client;

/**
 *
 * @author APDL
 * @version
 */
public class ExecutionAgent extends HttpServlet {

    /** Processes requests for both HTTP <code>GET</code> and <code>POST</code>
methods.
     * @param request servlet request
     * @param response servlet response
     */
    protected void processRequest(HttpServletRequest request,
HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet ExecutionAgent</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Servlet ExecutionAgent at " + request.getContextPath
() + "</h1>");

        /**
         * Database connection parameters
         */
        String dbUrl = System.getProperty("APDLdb.url",
"jdbc:derby://localhost:1527/APDLdb");
        String dbDriver = System.getProperty("APDLdb.driver",
"org.apache.derby.jdbc.ClientDriver");
        String dbUser = System.getProperty("APDLdb.user", "apdl");
        String dbPassword = System.getProperty("APDLdb.password", "apdl");

        /**
         * Connect to derby database
         **/
        Connection conn = null;
        try {
            Class.forName(dbDriver).newInstance();
            conn = DriverManager.getConnection(dbUrl, dbUser, dbPassword);
       } catch (Exception e) {
            out.println("<h3>" + e + "</h3>");
            out.println("<h3>Failed to load database driver!</h3>");
       }

        /**
         * Get process details from database
         **/
        ResultSet rs = null;
        try {
            String sql = "SELECT process_string FROM apdl.process_maps";
            PreparedStatement psGet = conn.prepareStatement(sql);
            rs = psGet.executeQuery();
       } catch (Exception e) {
            out.println("<h3>" + e + "</h3>");
            out.println("<h3>Failed to retrieve process maps from
database!</h3>");
       }

        /**
         * Store existing service information in an ArrayList
         **/
        ArrayList<String> maps = new ArrayList<String>();
        try {
            while (rs.next()) {
                String getMap = rs.getString("process_string");
                maps.add(getMap);
            }
            rs.close();
       } catch (Exception e) {
            out.println("<h3>" + e + "</h3>");
            out.println("<h3>Failed to store existing process
information!</h3>");
       }

        /**
         * Execute process
         */
        String process = maps.get(0);
        Pattern pStep = Pattern.compile("[;\\s]+");
        String[] steps = pStep.split(process);

        String input = request.getParameter("input");
        String output = null;
        for (int i=0; i<steps.length; i++) {
            Pattern pDetail = Pattern.compile(":");
            String[] details = pDetail.split(steps[i]);

            /**
             * Get web service wsdl locations from registry
             **/
            ResultSet rsWSDL = null;
            try {
                String sql = "SELECT wsdl_location FROM apdl.web_services WHERE
service_name = ?";
                PreparedStatement psGet = conn.prepareStatement(sql);
                psGet.setString(1, details[0]);
                rsWSDL = psGet.executeQuery();
            } catch (Exception e) {
                out.println("<h3>" + e + "</h3>");
                out.println("<h3>Failed to retrieve wsdl location from
registry!</h3>");
            }

            /**
             * Store wsdl location
             **/
            String wsdl = null;
            try {
                while (rsWSDL.next()) {
                    wsdl = rsWSDL.getString("wsdl_location");
                }
                rsWSDL.close();
            } catch (Exception e) {
                out.println("<h3>" + e + "</h3>");
                out.println("<h3>Failed to wsdl location!</h3>");
            }

            try {
                Client client = new Client(new URL(wsdl));
                Object[] servResults = client.invoke(details[1], new Object[]
{input});
                output = (String)servResults[0];
            } catch (Exception e) {
              out.println("<h3>" + e + "</h3>");
                out.println("<h3>Failed to access web service method!</h3>");
            }
            input = output;
            out.println("<h2>Step #" + (i+1) + " -</h2>");
            out.println("<h3>Service Name: " + details[0] + "</h3>");
            out.println("<h3>Operation Name: " + details[1]  + "</h3>");
            out.println("<h3>Input Type: " + details[2] + "</h3>");
        }

        /**
         * Close database connection
         */
        try {
            conn.close();
        } catch (Exception e) {
            out.println("<h3>" + e + "</h3>");
            out.println("<h3>Failed to close database connection!</h3>");
        }

        out.println("</body>");
        out.println("</html>");
        out.close();
    }

    // <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click
on the + sign on the left to edit the code.">
    /** Handles the HTTP <code>GET</code> method.
     * @param request servlet request
     * @param response servlet response
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

    /** Handles the HTTP <code>POST</code> method.
```

```
     * @param request servlet request
     * @param response servlet response
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse
response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

    /** Returns a short description of the servlet.
     */
    public String getServletInfo() {
        return "Short description";
    }
    // </editor-fold>
}
```

CATIAServiceSEI.java

```
package org.apdl.services;

/**
 * This is the service endpoint interface for the ServiceA web service.
 * Created Jun 4, 2006 7:43:53 PM
 * @author APDL
 */

public interface CATIAServiceSEI extends java.rmi.Remote {
    /**
     * Web service operation
     */
    public java.lang.String getServiceName() throws java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getServiceDescription() throws
java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getOperationName() throws java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getInputString() throws java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getOutputString() throws java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String CATIAUpdate(String input) throws
java.rmi.RemoteException;

}
```

CATIAServiceImpl.java

```
package org.apdl.services;

import java.sql.*;
import java.io.*;
import java.util.regex.Pattern;

/**
 * This is the implementation bean class for the ServiceA web service.
 * Created Jun 4, 2006 7:43:53 PM
 * @author APDL
 */
public class CATIAServiceImpl implements CATIAServiceSEI {
    /**
     * Default constructor for xfire
     */
      public CATIAServiceImpl() {};

    /**
     * Web service description parameters
     */
    private String serviceName = "CATIAService";
    private String serviceDescription = "This service updates the parametric
CAD model and outputs " +
                                        "iges files for the structural wedge
and air solid.";
    private String operationName = "CATIAUpdate";
    private String inputString = "nil";
    private String outputString = "iges";

    /**
     * Web service operation
     */
    public java.lang.String getServiceName() throws java.rmi.RemoteException {
        return serviceName;
    }

    /**
     * Web service operation
     */
    public java.lang.String getServiceDescription() throws
java.rmi.RemoteException {
        return serviceDescription;
    }

    /**
     * Web service operation
     */
    public java.lang.String getOperationName() throws java.rmi.RemoteException
{
        return operationName;
    }

    /**
     * Web service operation
     */
    public java.lang.String getInputString() throws java.rmi.RemoteException {
        return inputString;
    }
    /**
     * Web service operation
     */
    public java.lang.String getOutputString() throws java.rmi.RemoteException {
        return outputString;
    }

    /**
     * Web service operation
     */
    public java.lang.String CATIAUpdate(String input) throws
java.rmi.RemoteException {
        Pattern pInputs = Pattern.compile("[;\\s]+");
        String[] inputs = pInputs.split(input);

        String workingDir = inputs[0];
        int nBlades = Integer.parseInt(inputs[1]);
        double leadingAng = Double.parseDouble(inputs[2]);
        double trailingAng = Double.parseDouble(inputs[3]);
        double ang1 = (180.0 / nBlades) - 2.5;
        double ang2 = (180.0 / nBlades) + 2.5;
        double ang3 = (360.0 / nBlades);
        double ang4 = (360.0 / nBlades) - 5;

        /**
         * Create CATScript file
         */
        String path = "C:/Impeller/" + workingDir;
        try {
            File dir = new File(path);
            dir.mkdirs();
            File f = new File(path + "/CATIAUpdate.CATScript");
            FileOutputStream out = new FileOutputStream(f);
            PrintStream p = new PrintStream(out);
            p.println ("Language=\"VBSCRIPT\"");
            p.println ("Sub CATMain()");
            p.println
("CATIA.Documents.Open(\"C:\\Impeller\\ImpellerWedge.CATPart\")");
            p.format
("CATIA.ActiveDocument.Part.HybridBodies.Item(\"Geometrical
Set.1\").HybridShapes.Item(\"Sweep.1\").GetAngle(2).Value = %f\n", new Object[]
{new Double(leadingAng)});
            p.format
("CATIA.ActiveDocument.Part.HybridBodies.Item(\"Geometrical
Set.1\").HybridShapes.Item(\"Plane.3\").Angle.Value = %f\n", new Object[] {new
Double(trailingAng)});
            p.format
("CATIA.ActiveDocument.Part.HybridBodies.Item(\"Geometrical
Set.1\").HybridShapes.Item(\"Revolute.4\").BeginAngle.Value = %f\n", new
Object[] {new Double(ang1)});
            p.format
("CATIA.ActiveDocument.Part.HybridBodies.Item(\"Geometrical
Set.1\").HybridShapes.Item(\"Revolute.4\").EndAngle.Value = %f\n", new Object[]
{new Double(ang2)});
            p.format
("CATIA.ActiveDocument.Part.HybridBodies.Item(\"Geometrical
Set.1\").HybridShapes.Item(\"Revolute.5\").BeginAngle.Value = %f\n", new
Object[] {new Double(ang3)});
            p.format
("CATIA.ActiveDocument.Part.HybridBodies.Item(\"Geometrical
Set.1\").HybridShapes.Item(\"Volume Revolve.6\").BeginAngle.Value = %f\n", new
Object[] {new Double(ang3)});
```

```
            p.println ("CATIA.ActiveDocument.Part.Update");
            p.println ("CATIA.ActiveDocument.ExportData \"C:\\Impeller\\" +
workingDir + "\\StructWedge.igs\", \"igs\"");
            p.println ("CATIA.ActiveDocument.Selection.Add
CATIA.ActiveDocument.Part.HybridBodies.Item(\"Geometrical Set.1\")");
            p.println ("CATIA.ActiveDocument.Selection.VisProperties.SetShow
catVisPropertyNoShowAttr");
            p.println ("CATIA.ActiveDocument.Selection.Clear");
            p.println ("CATIA.ActiveDocument.Selection.Add
CATIA.ActiveDocument.Part.Bodies.Item(\"PartBody\")");
            p.println ("CATIA.ActiveDocument.Selection.VisProperties.SetShow
catVisPropertyShowAttr");
            p.println ("CATIA.ActiveDocument.Selection.Clear");
            p.format
("CATIA.ActiveDocument.Part.Bodies.Item(\"PartBody\").Shapes.Item(\"Shaft.1\").
FirstAngle.Value = %f\n", new Object[] {new Double(ang4)});
            p.println ("CATIA.ActiveDocument.Part.Update");
            p.println ("CATIA.ActiveDocument.ExportData \"C:\\Impeller\\" +
workingDir + "\\AirSolid.igs\", \"igs\"");
            p.println ("CATIA.ActiveDocument.Close");
            p.println ("CATIA.Quit()");
            p.println ("End Sub");
            p.close();
        } catch (Exception e) {
            e.printStackTrace();
        }

        try {
            String cmd[] = new String[3];
            cmd[0] = "cmd.exe";
            cmd[1] = "/C";
            cmd[2] = "C:\\Catia\\B15\\intel_a\\code\\bin\\CNEXT.exe -macro
C:\\Impeller\\" + workingDir + "\\CATIAUpdate.CATScript";

            Runtime rt = Runtime.getRuntime();
            Process proc = rt.exec(cmd);
            proc.waitFor();
        } catch (Exception e) {
            e.printStackTrace();
        }

        return workingDir;
    }

}
```

HyperMeshServiceSEI.java

```
package org.apdl.services;

/**
 * This is the service endpoint interface for the ServiceA web service.
 * Created Jun 4, 2006 7:43:53 PM
 * @author APDL
 */

public interface HyperMeshServiceSEI extends java.rmi.Remote {
    /**
     * Web service operation
     */
    public java.lang.String getServiceName() throws java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getServiceDescription() throws
java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getOperationName() throws java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getInputString() throws java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getOutputString() throws java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String HyperMeshUpdate(String input) throws
java.rmi.RemoteException;

}
```

HyperMeshServiceImpl.java

```
package org.apdl.services;

import java.sql.*;
import java.io.*;
import java.util.regex.Pattern;

/**
 * This is the implementation bean class for the ServiceA web service.
 * Created Jun 4, 2006 7:43:53 PM
 * @author APDL
 */
public class HyperMeshServiceImpl implements HyperMeshServiceSEI {
    /**
     * Default constructor for xfire
     */
      public HyperMeshServiceImpl() {};

    /**
     * Web service description parameters
     */
    private String serviceName = "HyperMeshService";
    private String serviceDescription = "This service creates a fluid mesh for
the air solid iges";
    private String operationName = "HyperMeshUpdate";
    private String inputString = "iges";
    private String outputString = "mesh";

    /**
     * Web service operation
     */
    public java.lang.String getServiceName() throws java.rmi.RemoteException {
        return serviceName;
    }

    /**
     * Web service operation
     */
    public java.lang.String getServiceDescription() throws
java.rmi.RemoteException {
        return serviceDescription;
    }

    /**
     * Web service operation
     */
    public java.lang.String getOperationName() throws java.rmi.RemoteException
{
        return operationName;
    }

    /**
     * Web service operation
     */
    public java.lang.String getInputString() throws java.rmi.RemoteException {
        return inputString;
    }

    /**
```

```
     * Web service operation
     */
    public java.lang.String getOutputString() throws java.rmi.RemoteException {
        return outputString;
    }

    /**
     * Web service operation
     */
    public java.lang.String HyperMeshUpdate(String input) throws
java.rmi.RemoteException {
         String workingDir = input;

        /**
         * Create HyperMesh command file
         */
        String path = "C:/Impeller/" + workingDir + "/HyperMeshUpdate.cmf";
        try {
            File f = new File(path);
            FileOutputStream out = new FileOutputStream(f);
            PrintStream p = new PrintStream(out);
            p.println ("*feinput(\"#iges\\iges\",\"C:/Impeller/" + workingDir +
"/AirSolid.igs\",1,0,-0.01,1,1)");
            p.println ("*collectorcreate(components,\"hub\",\"default\",8)");
            p.println
("*collectorcreate(components,\"pressure\",\"default\",10)");
            p.println
("*collectorcreate(components,\"suction\",\"default\",12)");
            p.println
("*collectorcreate(components,\"inlet\",\"default\",13)");
            p.println
("*collectorcreate(components,\"outlet\",\"default\",6)");
            p.println
("*collectorcreate(components,\"shroud\",\"default\",15)");
            p.println ("*collectorcreate(components,\"fluid\",\"default\",1)");
            p.println ("*currentcollector(components,\"hub\")");
            p.println ("*createmark(surfaces,1) 1");
            p.println ("*defaultmeshsurf(1,0.75,0,0,1,0,0,0,1,0,0,0,0)");
            p.println ("*currentcollector(components,\"outlet\")");
            p.println ("*createmark(surfaces,1) 2");
            p.println ("*defaultmeshsurf(1,0.75,0,0,1,0,0,0,1,0,0,0,0)");
            p.println ("*currentcollector(components,\"shroud\")");
            p.println ("*createmark(surfaces,1) 3");
            p.println ("*defaultmeshsurf(1,0.75,0,0,1,0,0,0,1,0,0,0,0)");
            p.println ("*currentcollector(components,\"inlet\")");
            p.println ("*createmark(surfaces,1) 4");
            p.println ("*defaultmeshsurf(1,0.75,0,0,1,0,0,0,1,0,0,0,0)");
            p.println ("*currentcollector(components,\"pressure\")");
            p.println ("*createmark(surfaces,1) 5");
            p.println ("*defaultmeshsurf(1,0.75,0,0,1,0,0,0,1,0,0,0,0)");
            p.println ("*currentcollector(components,\"suction\")");
            p.println ("*createmark(surfaces,1) 6");
            p.println ("*defaultmeshsurf(1,0.75,0,0,1,0,0,0,1,0,0,0,0)");
            p.println ("*createmark(elements,1) \"all\"");
            p.println ("*equivalence(elements,1,0.01,1,0,0)");
            p.println ("*currentcollector(components,\"fluid\")");
            p.println ("*createmark(elements,1) \"all\"");
            p.println ("*createmark(components,2)");
            p.println ("*tetramesh(elements,1,components,2,1.2,0.75,1)");
            p.println ("*createpoint(0.0,0.0,0.0,0)");
            p.println
("*templatefileset(\"C:/Altair/hw7.0/templates/feoutput/nastran/general\")");
```

```
            p.println
("*feoutput(\"C:/Altair/hw7.0/templates/feoutput/nastran/general\",\"C:/Impelle
r/" + workingDir + "/HyperToFluent.nas\",1,0,0)");
            p.println ("*writefile(\"C:/Impeller/" + workingDir +
"/AirMesh.hm\",1)");
            p.println ("*quit(1)");
            p.close();
        } catch (Exception e) {
            e.printStackTrace();
        }

        try {
            String cmd[] = new String[3];
            cmd[0] = "cmd.exe";
            cmd[1] = "/C";
            cmd[2] = "C:\\Altair\\hw7.0\\hm\\bin\\hmopengl.exe -
cC:\\Impeller\\" + workingDir + "\\HyperMeshUpdate.cmf";

            Runtime rt = Runtime.getRuntime();
            Process proc = rt.exec(cmd);
            proc.waitFor();
        } catch (Exception e) {
            e.printStackTrace();
        }

        return workingDir;
    }

}
```

FluentServiceSEI.java

```
package org.apdl.services;

/**
 * This is the service endpoint interface for the ServiceA web service.
 * Created Jun 4, 2006 7:43:53 PM
 * @author APDL
 */

public interface FluentServiceSEI extends java.rmi.Remote {
    /**
     * Web service operation
     */
    public java.lang.String getServiceName() throws java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getServiceDescription() throws
java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getOperationName() throws java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getInputString() throws java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getOutputString() throws java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String FluentUpdate(String input) throws
java.rmi.RemoteException;

}
```

FluentServiceImpl.java

```
package org.apdl.services;

import java.sql.*;
import java.io.*;
import java.util.regex.Pattern;

/**
 * This is the implementation bean class for the ServiceA web service.
 * Created Jun 4, 2006 7:43:53 PM
 * @author APDL
 */
public class FluentServiceImpl implements FluentServiceSEI {
    /**
     * Default constructor for xfire
     */
      public FluentServiceImpl() {};

    /**
     * Web service description parameters
     */
    private String serviceName = "FluentService";
    private String serviceDescription = "This service solves for the impeller
pressures.";
    private String operationName = "FluentUpdate";
    private String inputString = "mesh";
    private String outputString = "pressure";

    /**
     * Web service operation
     */
    public java.lang.String getServiceName() throws java.rmi.RemoteException {
        return serviceName;
    }

    /**
     * Web service operation
     */
    public java.lang.String getServiceDescription() throws
java.rmi.RemoteException {
        return serviceDescription;
    }

    /**
     * Web service operation
     */
    public java.lang.String getOperationName() throws java.rmi.RemoteException
{
        return operationName;
    }

    /**
     * Web service operation
     */
    public java.lang.String getInputString() throws java.rmi.RemoteException {
        return inputString;
    }

    /**
```

```
     * Web service operation
     */
    public java.lang.String getOutputString() throws java.rmi.RemoteException {
        return outputString;
    }

    /**
     * Web service operation
     */
    public java.lang.String FluentUpdate(String input) throws
java.rmi.RemoteException {
         String workingDir = input;

        /**
         * Create Fluent command file
         */
        String path = "C:/Impeller/" + workingDir;
        try {
            File f = new File(path + "/FluentUpdate.log");
            FileOutputStream out = new FileOutputStream(f);
            PrintStream p = new PrintStream(out);
            p.println ("file import nastran C:\\Impeller\\" + workingDir +
"\\HyperToFluent.nas");
            p.println ("grid scale\n0.025\n0.025\n0.025");
            p.println ("define units pressure\natm");
            p.println ("define units angular-velocity\nrpm");
            p.println ("define models solver\nc-i\ny\nquit");
            p.println ("define models energy\ny");
            p.println ("define models viscous ke-realizable\ny");
            p.println ("define materials change-create air\nair\ny\nideal-
gas\nn\nn\nn\nn\nn\nn\nn\nn");
            p.println ("define o-c o-p\n0");
            p.println ("define\nb-c\nm-z\nz-t\noutlet\np-o\nz-t\ninlet\np-
i\nquit");
            p.println ("p-i
inlet\nn\n100000\nn\n1\nn\n50\ny\nn\ny\nn\n0\nn\n0\nn\n1\ny\nn\n10\nn\n10");
            p.println ("p-o
outlet\nn\n1.8014\nn\nn\n300\nn\ny\nn\nn\ny\n10\n10");
            p.println ("wall hub\n0\nn\n0\nn\nn\nn\n0\nn\nn\nn\n0\n0.5");
            p.println ("wall pressure\n0\nn\n0\nn\nn\nn\n0\nn\nn\nn\n0\n0.5");
            p.println ("wall suction\n0\nn\n0\nn\nn\nn\n0\nn\nn\nn\n0\n0.5");
            p.println ("wall shroud\n0\nn\n0\nn\nn\nn\n0\nn\ny\nm-b-
m\nn\ny\nn\n0\n0.5\n0\n0\n0\n0\n0\n0\n1");
            p.println ("fluid fluid\ny\nair\nn\nn\nn\ny\n0\n0\n0\n-
20000\n0\n0\n0\n0\n0\n1\nn\nn\nn\nquit\nquit");
            p.println ("solve initialize\nset-
defaults\np\n100000\nx\n0\ny\n0\nz\n100\nquit\nquit");
            p.println ("solve set\nc\n0.001\nunder-
relaxation\nepsilon\n0.4\nk\n0.4\nt-v\n0.5\nsolid\n0.5\nquit\nd-s\namg-
c\n0\nk\n0\nepsilon\n0\nquit\nquit");
            p.println ("solve monitors residual\nplot\ny\nquit");
            p.println ("solve iterate\n65");
            p.println ("display\nset\nhard-
copy\ncolor\ncolor\nquit\ndriver\njpeg\nquit\nquit\nquit");
            p.println ("hard-copy\nC:\\Impeller\\" + workingDir +
"\\residuals.jpg\ny");
            p.println ("vector\nvelocity\nv-m\n\n\n\n\nquit");
            p.println ("view\nread-view C:\\Impeller\\velocity\nrestore-view
good\nquit");
            p.println ("display\nhard-copy\nC:\\Impeller\\" + workingDir +
"\\velocity.jpg\ny\nquit");
            p.println ("file\nexport\nascii C:\\Impeller\\" + workingDir +
"\\Pressure.txt\npressure\n\nn\nn\npressure\nnone\nn\ny");
```

```
            p.println ("ascii C:\\Impeller\\" + workingDir +
"\\Suction.txt\nsuction\n\nn\nn\npressure\nnone\nn\ny");
            p.println ("ascii C:\\Impeller\\" + workingDir +
"\\Hub.txt\nhub\n\nn\nn\npressure\nnone\nn\ny");
            p.println ("quit\nquit\nexit\ny");
            p.close();
        } catch (Exception e) {
            e.printStackTrace();
        }

        try {
            String cmd[] = new String[3];
            cmd[0] = "cmd.exe";
            cmd[1] = "/C";
            cmd[2] = "C:\\Fluent\\ntbin\\ntx86\\fluent 3d -i C:\\Impeller\\" +
workingDir + "\\FluentUpdate.log";

            Runtime rt = Runtime.getRuntime();
            Process proc = rt.exec(cmd);

            File fOut = new File(path + "/Hub.txt");
            while (!fOut.exists()) {
                Thread.sleep(10000);
            }

        } catch (Exception e) {
            e.printStackTrace();
        }

        return workingDir;
    }

}
```

ANSYSServiceSEI.java

```
package org.apdl.services;

/**
 * This is the service endpoint interface for the ServiceA web service.
 * Created Jun 4, 2006 7:43:53 PM
 * @author APDL
 */

public interface ANSYSServiceSEI extends java.rmi.Remote {
    /**
     * Web service operation
     */
    public java.lang.String getServiceName() throws java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getServiceDescription() throws
java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getOperationName() throws java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getInputString() throws java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String getOutputString() throws java.rmi.RemoteException;

    /**
     * Web service operation
     */
    public java.lang.String ANSYSUpdate(String input) throws
java.rmi.RemoteException;
}
```

ANSYSServiceImpl.java

```
package org.apdl.services;

import java.sql.*;
import java.io.*;
import java.util.ArrayList;
import java.util.regex.Pattern;

/**
 * This is the implementation bean class for the ServiceA web service.
 * Created Jun 4, 2006 7:43:53 PM
 * @author APDL
 */
public class ANSYSServiceImpl implements ANSYSServiceSEI {
    /**
     * Default constructor for xfire
     */
      public ANSYSServiceImpl() {};

    /**
     * Web service description parameters
     */
    private String serviceName = "ANSYSService";
    private String serviceDescription = "This service solves for the impeller
stresses.";
    private String operationName = "ANSYSUpdate";
    private String inputString = "pressure";
    private String outputString = "stress";

    /**
     * Web service operation
     */
    public java.lang.String getServiceName() throws java.rmi.RemoteException {
        return serviceName;
    }

    /**
     * Web service operation
     */
    public java.lang.String getServiceDescription() throws
java.rmi.RemoteException {
        return serviceDescription;
    }

    /**
     * Web service operation
     */
    public java.lang.String getOperationName() throws java.rmi.RemoteException
{
        return operationName;
    }

    /**
     * Web service operation
     */
    public java.lang.String getInputString() throws java.rmi.RemoteException {
        return inputString;
    }
    /**
     * Web service operation
     */
    public java.lang.String getOutputString() throws java.rmi.RemoteException {
        return outputString;
    }

    /**
     * Web service operation
     */
    public java.lang.String ANSYSUpdate(String input) throws
java.rmi.RemoteException {
        String workingDir = input;

        /**
         * Parse Fluent output and determine average pressures
         */
        String pressureFile = "C:/Impeller/" + workingDir + "/Pressure.txt";
        String suctionFile = "C:/Impeller/" + workingDir + "/Suction.txt";
        String hubFile = "C:/Impeller/" + workingDir + "/Hub.txt";

        double pressure = parsePressure(pressureFile);
        double suction = parsePressure(suctionFile);
        double hub = parsePressure(hubFile);

        /**
         * Create ANSYS batch file
         */
        String path = "C:/Impeller/" + workingDir;
        try {
            File f = new File(path + "/ANSYSUpdate.mac");
            FileOutputStream out = new FileOutputStream(f);
            PrintStream p = new PrintStream(out);
            p.println
("/AUX15\nIOPTN,IGES,NODEFEAT\nIOPTN,MERGE,YES\nIOPTN,SOLID,YES\nIOPTN,SMALL,YE
S\nIOPTN,GTOLER,DEFA\nIGESIN,'StructWedge','igs'\nVPLOT");
            p.println
("/PREP7\nET,1,SOLID187\nMPTEMP,,,,,,,,\nMPTEMP,1,0\nMPDATA,EX,1,,2e11\nMPDATA,
PRXY,1,,.3\nMPTEMP,,,,,,,,\nMPTEMP,1,0\nMPDATA,DENS,1,,7800");
            p.println ("MSHAPE,1,3D\nMSHKEY,0\nCM,_Y,VOLU\nVSEL, , ,
,1\nCM,_Y1,VOLU\nCHKMSH,'VOLU'\nCMSEL,S,_Y\nVMESH,_Y1\nCMDELE,_Y\nCMDELE,_Y1\nC
MDELE,_Y2");
            p.format ("SFA,P51X,1,PRES,%f\n", new Object[] {new
Double(pressure)});
            p.println ("FLST,2,1,5,ORDE,1\nFITEM,2,15");
            p.format ("SFA,P51X,1,PRES,%f\n", new Object[] {new
Double(suction)});
            p.println ("FLST,2,1,5,ORDE,1\nFITEM,2,19");
            p.format ("SFA,P51X,1,PRES,%f\n", new Object[] {new Double(hub)});
            p.println ("FLST,2,2,5,ORDE,2\nFITEM,2,22\nFITEM,2,-23");
            p.println ("DA,P51X,ALL,0\nOMEGA,0,0,-20000,0\nFINISH");
            p.println ("/SOL\n/STATUS,SOLU\nSOLVE\nFINISH");
            p.println
("/POST1\n/COLOR,WBAK,WHIT,1\n/SHOW,JPEG\nJPEG,QUAL,75,\nJPEG,ORIENT,HORIZ\nJPE
G,COLOR,2\nJPEG,TMOD,1\n/GFILE,1600,");
            p.println ("/VIEW,  1, -0.729260062836    , -0.657795420059    ,
0.188374483681");
            p.println ("/ANG,   1,   80.3946751086");
            p.println ("/EFACET,1\nPLNSOL,S,EQV,0,1.0");
            p.println ("/FOC,   1,  -9.80127937324    ,   8.54300250559    ,
10.1371984318");
            p.println ("/VIEW,  1, -0.517366709999E-01,  0.925379003517    ,
0.375495694681");
```

```
            p.println ("/ANG,   1,   174.605743318");
            p.println ("/EFACET,1\nPLNSOL,S,EQV,0,1.0\nPRNSOL,S,PRIN");
            p.close();
        } catch (Exception e) {
            e.printStackTrace();
        }

        try {

            String ANSYSCommand = "\"C:\\Program Files\\Ansys 
Inc\\v81\\ANSYS\\bin\\Intel\\ansys.exe\"  -p ANSYSRF -dir " +
                             "\"C:\\Impeller\\" + workingDir + "\" -j \"ANSYS\" 
-s read -l en-us -b " +
                             "< \"C:\\Impeller\\" + workingDir + 
"\\ANSYSUpdate.mac\" > " +
                             "\"C:\\Impeller\\" + workingDir + 
"\\ANSYSOutput.out\"";
            String ANSYSPath = path + "/ANSYSCmd.bat";

            File bf = new File(ANSYSPath);
            FileOutputStream bout = new FileOutputStream(bf);
            PrintStream bp = new PrintStream(bout);
            bp.println (ANSYSCommand);
            bp.close();

            String cmd[] = new String[3];
            cmd[0] = "cmd.exe";
            cmd[1] = "/C";
            cmd[2] = ANSYSPath;

            Runtime rt = Runtime.getRuntime();
            Process proc = rt.exec(cmd);
            proc.waitFor();
        } catch (Exception e) {
            e.printStackTrace();
        }

        return workingDir;
    }

    /**
     * Method for parsing Fluent output files and determining the average 
pressures
     */
    private double parsePressure(String fileName) {
        FileReader inFile = null;
        BufferedReader in = null;
        String line;
        ArrayList<Double> values = new ArrayList<Double>();
        double avg = 0.0;

        try {
            inFile = new FileReader(fileName);
            in = new BufferedReader(inFile);

            // Parse the file header
            String header = in.readLine().trim();

            // Parse the node pressures
            while ((line = in.readLine()) != null) {
                line = line.trim();
                if (line.length() != 0) {
                    Pattern delimiters = Pattern.compile("[;\\s]+");
                    String[] tokens = delimiters.split(line);
                    values.add(Double.parseDouble(tokens[4]));
                }
            }

            // Reorder the pressure data in increasing order
            int j = 0;
            int count = 1;
            double index;
            for (int i=1; i < values.size(); i++) {
                index = values.get(i);
                j = count;

                while ((j > 0) && (values.get(j-1) > index)) {
                    values.set(j, values.get(j-1));
                    j = j-1;
                }

                values.set(j, index);

                count++;
            }

            // Remove outliers from the data and compute the average
            double median;
            if (values.size()%2 == 0) {
                median = (values.get(values.size()/2) +
values.get((values.size() + 2)/2))/2;
            } else {
                median = values.get((values.size() + 1)/2);
            }

            double sum = 0;
            count = 0;
            for (int i=0; i < values.size(); i++) {
                if (values.get(i) > 0 && values.get(i) < (3 * median)) {
                    sum += values.get(i);
                    count++;
                }
            }

            avg = sum/count;

            in.close();

        } catch (Exception e) {
          e.printStackTrace();
        }

        return avg;
    }

}
```