J Intell Manuf (2008) 19:537–552
DOI 10.1007/s10845-008-0136-y

A dynamic workflow framework for mass customization using web service and autonomous agent techniques

Daniel J. Karpowitz · Jordan J. Cox · Jeffrey C. Humpherys · Sean C. Warnick

Accepted: 17 May 2007 / Published online: 18 June 2008

Abstract 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.

Keywords Autonomous agents · Product development · WEB services · Ontologies · Automation

Introduction

Tseng and Jiao define mass customization as "producing goods and services to meet individual customer's needs with near mass production efficiency" (Tseng and Jiao 2001). 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 (Pine 1993). 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 (Lander 1997).

This work presents a potential solution to reduce the risk associated with the custom software development investment.

D. J. Karpowitz · J. J. Cox (✉)
Department of Mechanical Engineering, Brigham Young University, 455 CTB, Provo, UT 84602, USA
e-mail: cox@byu.edu

D. J. Karpowitz
e-mail: daniel.j.karpowitz@boeing.com

J. C. Humpherys
Department of Mathematics, Brigham Young University, Provo, UT, USA

S. C. Warnick
Department of Computer Science, Brigham Young University, Provo, UT, USA

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 paper 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.

Background

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" (Roach et al. 2005). 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 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 (Roach et al. 2005).

The development of the PDG and the mapping of individual engineering processes require the creation of a Product Transformation Schematic (PTS) within a specified envelope of variation. The PTS is defined by 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 (Roach 2003).

Service-oriented architecture (SOA)

A service-oriented architecture (SOA) is another 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 (Ort 2005). 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. 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") (Young 2005). 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 (Ort 2005).

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.

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.

Web service standards

Web services provide an effective, standards-based approach to SOA. Each of these standards is based on eXtensible Markup Language (XML), a general purpose markup language, which uses "tags" similar to those used by HTML, for describing and exchanging data over a network in an organized and structured way.

The core web service standards defined in the WS-I basic profile (Ballinger et al. 2006) include: Simple Object Access Protocol (SOAP), an XML-based protocol for "exchanging structured and typed information between peers in a decentralized, distributed environment" (Mitra 2003); Web Service Description Language (WSDL), an XML approach to defining how to communicate with a given web service; and Universal Description, Discovery, and Integration (UDDI), which defines "how to publish and discover information about services in a UDDI-conforming registry" (Ort 2005). 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.

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 (Wooldridge and Jennings 1998). 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" (Lander 1997). Such extensive change makes software maintenance in engineering design a particularly complex and difficult task. It is estimated that traditionally, software maintenance consumes 50–80% of an application's lifecycle cost (Lander 1997). 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" (Lander 1997). 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 (Feng 2005). These flexible characteristics dramatically reduce the custom software development and future maintenance associated with traditional automated engineering systems.

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" (Wooldridge 1999). 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 (Wooldridge and Jennings 1998; Wooldridge 1999; Wooldridge and Dickinson 2005). Despite these discrepancies, it is generally accepted that agent autonomy is key to understanding this type of agency (Wooldridge and Jennings 1995).

Wooldridge makes several important points about agent definition (Wooldridge and Jennings 1995). 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:

- Autonomous—Agents control both their individual state and behavior.
- Reactive—Agents are able to perceive changes in their environment and respond in a timely manner.
- Proactive—Agents demonstrate goal-oriented behavior by taking initiative to meet objectives.
- Social—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 (Wooldridge and Jennings 1995). 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 (Wooldridge and Jennings 1995). Russell and Norvig (Russell and Norvig 1995) 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.

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 (Shen et al. 2001). 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) (FIPA 2002). 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.

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" (Bray et al. 2006). 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" (Wooldridge and Dickinson 2005). 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" (Greenwood and Calisti 2004). Rather, agents "offer multiple services, or behaviors,

that can be processed concurrently and activated specifying goals" (Greenwood and Calisti 2004). 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 (JADE 2006). 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 (FIPA 2006). 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). 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" (Greenwood and Calisti 2004).

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 (Wooldridge and Dickinson 2005). 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 (Wooldridge and Dickinson 2005). 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 (Wooldridge and Dickinson 2005).

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 (Wooldridge and Dickinson 2005).

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 (Maximilien and Singh 2003); 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 (Buhler et al. 2003); 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 (Laukkanen and Helin 2003); 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 (Sycara et al. 2004).

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 (McIlraith et al. 2001).

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" (Hendler 2001). 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 (Shen et al. 2001).

Web service and autonomous agent systems benefit from using ontologies to "decipher the content of exchanged messages" (Shen et al. 2001). 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 (Microsoft 2006) 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) (Martin 2004) 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) (Farrell and Lausen 2007), OWL-S is an attempt to represent meaning and semantics with machine-interpretable content.

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 (Shen et al. 2001). For many systems, particularly the dynamic product development framework presented in this paper, a simplified, less-flexible approach to ontology is sufficient.

Methods

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:

1. Managing a web service registry and identifying potential services.
2. Configuring process maps of all potential workflow paths.
3. 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.

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 (Martin 2004) 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.

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 (Roach et al. 2005). 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. One procedure for identifying links between the process input and output sets is presented by Laukkanen and Helin (Laukkanen and Helin 2003). By using a backwards mapping technique, 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 (Andrews et al. 2003). 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 (Laukkanen and Helin 2003); 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.

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.

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. (Roach et al. 2005) 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 (Greenwood and Calisti 2004).

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.

Results

Case study no. 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 Fig. 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.

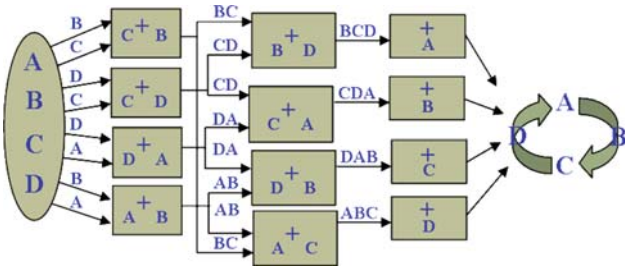

Fig. 1 Process schematic for creation of basic ring structure

The schematic shown in Fig. 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 most basic and unique functions need to be captured as services.

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 (XFire 2006) 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 2 and 3 show the MA adding the available web services and updating their descriptive information in this two-step process.

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

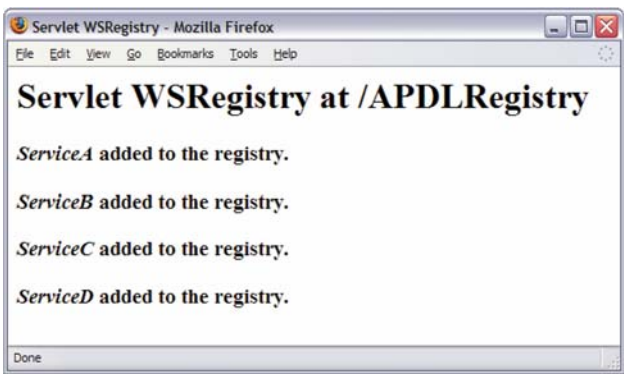

Fig. 2 Registered web services from "services.xml" file

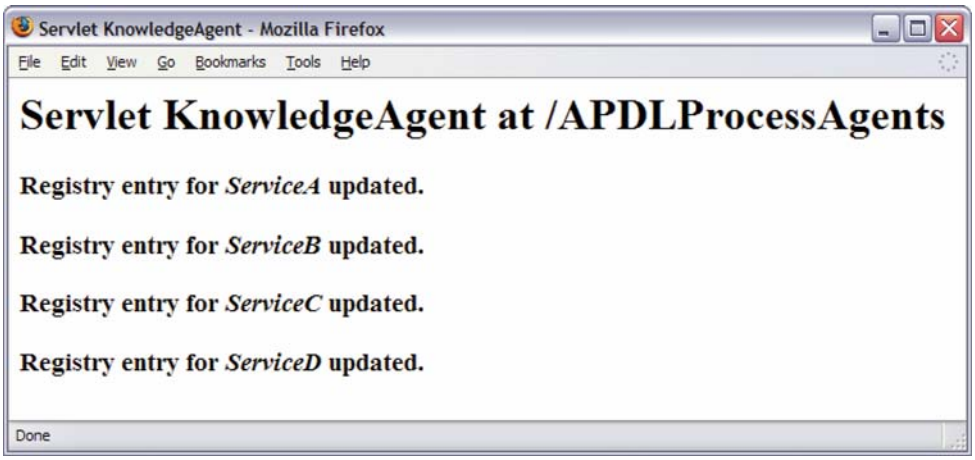

Fig. 3 Registry update from web service reporting methods

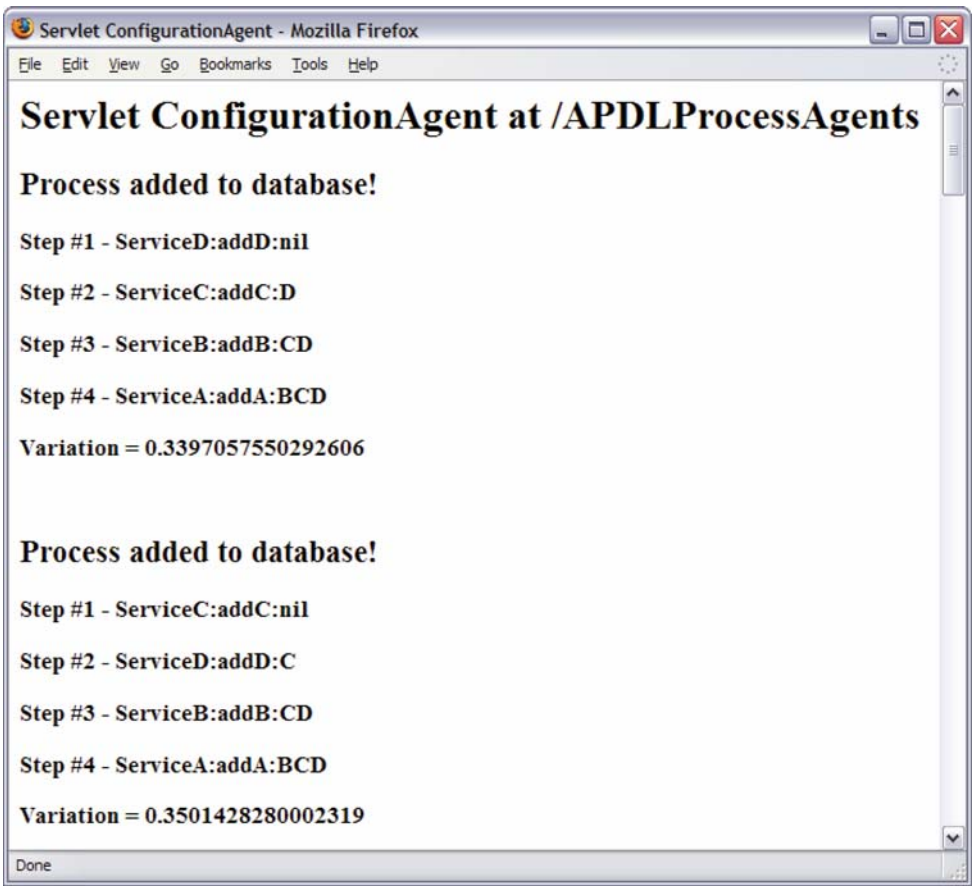

Fig. 4 Example workflow paths mapped by configuration agent

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 16 available processes, two of which are shown in Fig. 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 shows the successful execution of the optimal workflow path based on this criterion.

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

Fig. 5 Workflow execution by process execution agent

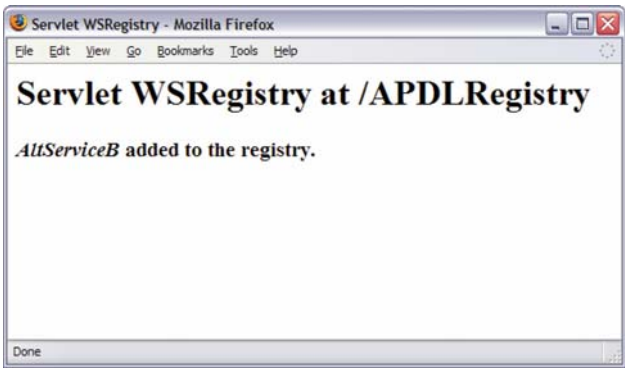

Fig. 6 Registry update with alternative web service

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 Figs. 6 and 7.

Fig. 7 Workflow execution with alternative web service included

Case study no. 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.

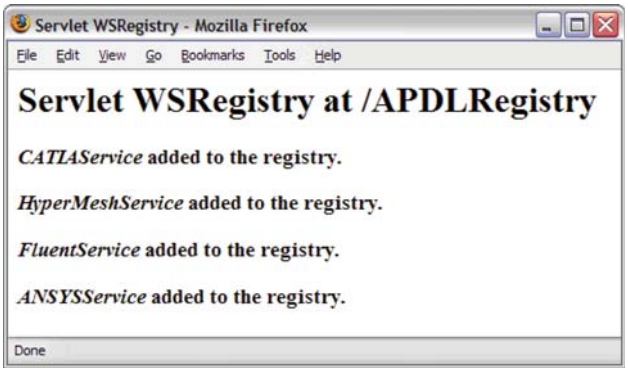

Fig. 8 Registered web services for impeller design process

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 8 shows the MA adding the process tasks to this database and Fig. 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.

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 10 shows the actual IGES data for the structural wedge

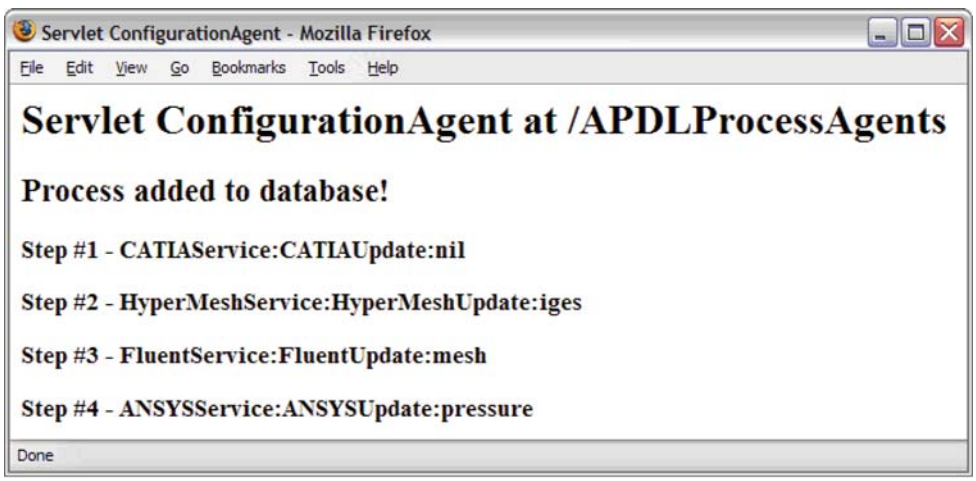

Fig. 9 Workflow path mapped by the configuration agent

Fig. 10 IGES data for the impeller structural wedge

StructWedge.igs - WordPad

File Edit View Insert Format Help

```
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
```

For Help, press F1

Fig. 11 CATIA representation of the structural wedge

while Fig. 11 shows the CATIA V5 interpretation of this information.

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 12 shows the surface mesh for the air solid wedge created 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 13 is a screenshot of the pressure distribution that is also output to the working directory by this service.

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 14 shows the Von Mises stress plot that is part of the service output.

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

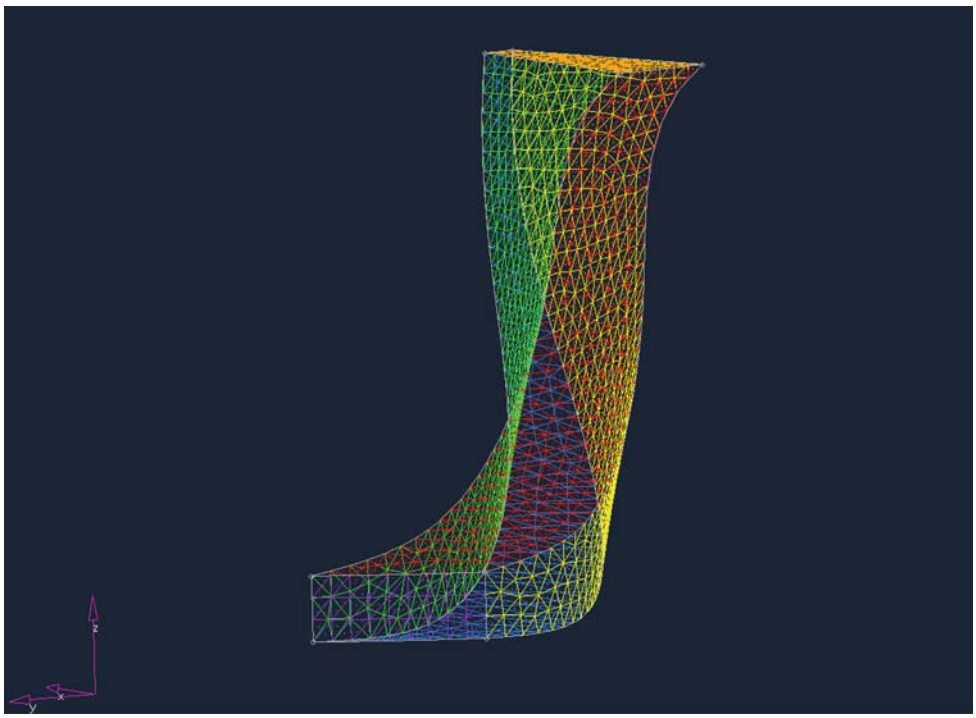

Fig. 12 Surface mesh for the air solid wedge produced by HyperMesh

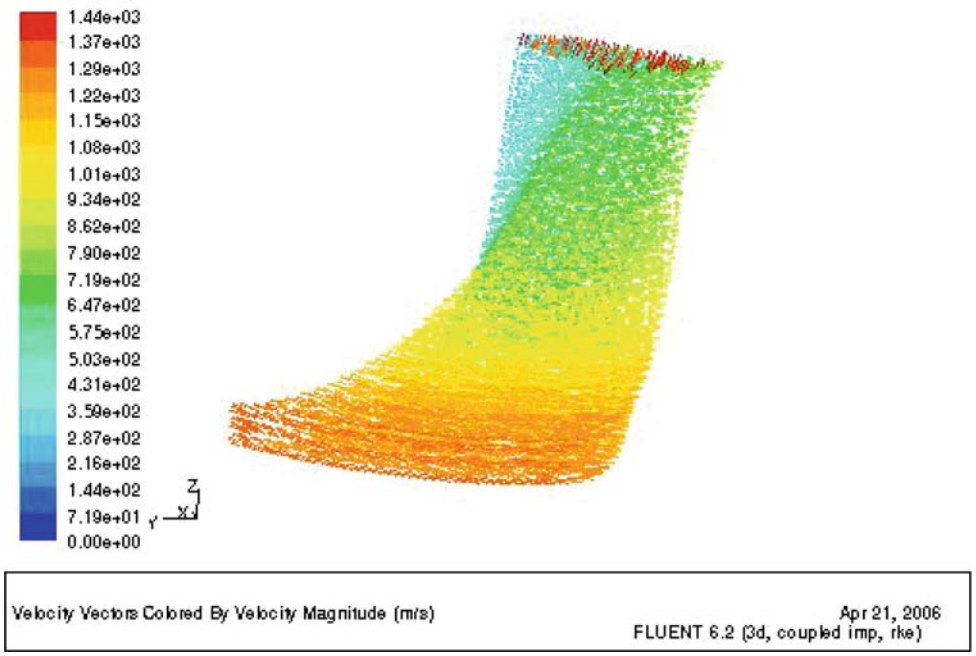

Fig. 13 Fluent pressure distribution for the air solid wedge

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 (Young 2005) without changing the methods already presented.

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

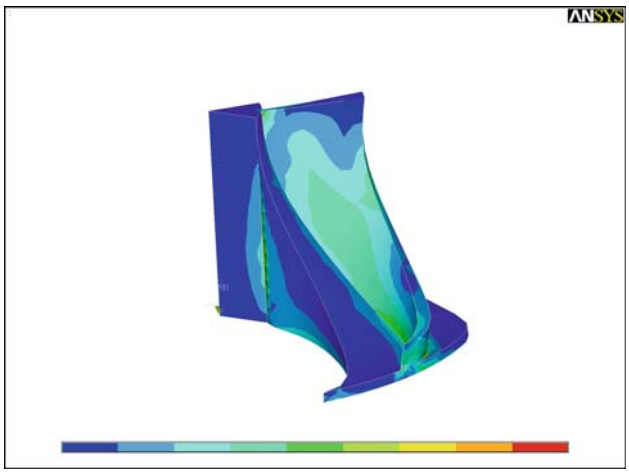

Fig. 14 Stress plot produced by ANSYS for the structural wedge

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.

References

Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., & Weerawarana, S. (2003). *Business process execution language for web services*. ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf, IBM Developer Works.

Ballinger, K., Ehnebuske, D., Ferris, C., Gudgin, M., Liu, C. K., Nottingham, M., & Yendluri, P. (Eds.). (2006). *WS-I basic profile version 1.1*. http://www.ws-i.org/Profiles/BasicProfile-1.1.html, Web Services Interoperability Organization.

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., Yergeau, F., & Cowan, J. (Eds.). (2006). *Extensible Markup Language (XML) 1.1*. http://www.w3.org/TR/xml11/, W3C.

Buhler, P., Vidal, J. N., & Verhagen, H. (2003). Adaptive workflow = web services + agents. In *Proceedings of the International Conference on Web Services*, Las Vegas, NV. CSREA Press.

Farrell, J., & Lausen, H. (Eds.). (2007). *Semantic annotations for WSDL*. http://www.w3.org/TR/sawsdl/, W3C.

Feng, S. C. (2005). Preliminary design and manufacturing planning integration using web-based intelligent agents. *Journal of Intelligent Manufacturing, 16*, 423–437.

Foundation for Intelligent Physical Agents (FIPA). (2002). *FIPA ACL message structure specification*. http://www.fipa.org/specs/fipa00061/index.html.

Foundation for Intelligent Physical Agents (FIPA). (2006). *Welcome to FIPA!* http://www.fipa.org/.

Greenwood, D., & Calisti, M. (2004). Engineering web service-agent integration. In *IEEE Systems, Cybernetics and Man Conference*, The Hague, Netherlands.

Hendler, J. (2001). Agents and the semantic web. In *IEEE Intelligent Systems*. Mar–Apr, 2001.

Java Agent DEvelopment Framework (JADE). (2006). *An open source platform for peer-to-peer agent based applications*. http://jade.tilab.com/.

Lander, S. E. (1997). Issues in multi-agent design systems. *IEE Expert, 12*(2), 8–26.

Laukkanen, M., & Helin, H. (2003). Composing workflows of semantic web services. In *Proceedings of the 1st International Workshop on Web Services and Agent Based Engineering*, Sydney, Australia.

Martin, D. (Ed.). (2004). *OWL-S: Semantic markup for web services*. http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/, W3C.

Maximilien, E. M., & Singh, M. P. (2003). Agent-based architecture for autonomic web service selection. In *Proceedings of the 1st International Workshop on Web Services and Agent Based Engineering*, Sydney, Australia.

McIlraith, S. A., Son, T.C., & Honglei, Z. (2001). Semantic web services. In *IEEE Intelligent Systems*, Mar–Apr, 2001.

Microsoft. (2006). *UDDI business registry shutdown FAQ*. http://uddi.microsoft.com/about/FAQshutdown.htm.

Mitra, N. (Ed.). (2003). *SOAP Version 1.2 Part 0: Primer*. http://www.w3.org/TR/soap12-part0/, W3C.

Ort, E. (2005). *Service-oriented architecture and web services: Concepts, technologies, and tools*. http://java.sun.com/developer/technicalArticles/WebServices/soa2/index.html, Sun Developer Network.

Pine, J. B. (1993). *Mass customization: The new frontier in business competition*. Boston: Harvard Business School Press.

Roach, G. M. (2003). *The product design generator—a next generation approach to detailed design*. Provo, UT: Brigham Young University.

Roach, G. M., Cox, J. J., & Sorenson, C. D. (2005). The product design generator: A system for producing design variants. *International Journal of Mass Customization, 1*(1), 83–106.

Russell, S., & Norvig, P. (1995). *Artificial intelligence: A modern approach*. Prentice-Hall.

Shen, W., Norrie, D. H., & Barthès, J. A. (2001). *Mult-agent systems for concurrent intelligent design and manufacturing*. London, England: Taylor and Francis.

Sycara, K., Paolucci, M., Soudry, J., & Srinivasan, N. (2004). Dynamic discovery and coordination of agent based semantic web services. In *IEEE Internet Computing*, May–June, 2004.

Tseng, M. M., & Jiao, J. (2001). Mass customization. In G. Slavendy (Ed.), *Handbook of Industrial Engineering: Technology and Operations Management* (3rd ed.). Hoboken, NJ: Wiley.

Wooldridge, M. (1999). Intelligent agents. In G. Weiss (Ed.), *Multiagent Systems*. Boston: The MIT Press.

Wooldridge, M., & Dickinson, I. (2005). 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)*, The Hague, Netherlands.

Wooldridge, M., & Jennings, N. R. (1995). Intelligent agents: Theory and practice. *Knowledge Engineering Review, 10*(2), 115–152.

Wooldridge, M., & Jennings, N. R. (1998). Pitfalls of agent-oriented development. In *Proceedings of the Second International Conference on Autonomous Agents*, Minneapolis, MN, pp. 385–391.

XFire. (2006). Codehaus XFire. http://xfire.codehaus.org/.

Young, J. M. (2005). *Nesting automated design modules in an interconnected framework*. Provo, UT: Brigham Young University.