Brigham Young University
BYU ScholarsArchive

Theses and Dissertations

2007-07-12

Autonomic Product Development Process Automation

John E. Daley
Brigham Young University - Provo

Follow this and additional works at: https://scholarsarchive.byu.edu/etd

 Part of the Mechanical Engineering Commons

BYU ScholarsArchive Citation

Daley, John E., "Autonomic Product Development Process Automation" (2007). *Theses and Dissertations*. 965.
https://scholarsarchive.byu.edu/etd/965

This Thesis is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.

AUTONOMIC PRODUCT DEVELOPMENT PROCESS AUTOMATION

by

John Daley

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

August 2007

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

John Daley

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	Christopher A. Mattson

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of John Daley in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date

Jordan J. Cox
Chair, Graduate Committee

Accepted for the Department

Matthew R. Jones
Graduate Coordinator

Accepted for the College

Alan R. Parkinson
Dean, Ira A. Fulton College of Engineering and Technology

ABSTRACT

AUTONOMIC PRODUCT DEVELOPMENT PROCESS AUTOMATION

John Daley

Department of Mechanical Engineering

Master of Science

Market globalization and mass customization requirements are forcing companies towards automation of their product development processes. Many task-specific software solutions provide localized automation. Coordinating these local solutions to automate higher-level processes requires significant software maintenance costs due to the incompatibility of the software tools and the dynamic nature of the product development environment. Current automation methods do not provide the required level of flexibility to operate in this dynamic environment.

An autonomic product development process automation strategy is proposed in order to provide a flexible, standardized approach to product development process automation and to significantly reduce the software maintenance costs associated with traditional automation methods. Key elements of the strategy include a formal approach to decompose product development processes into services, a method to

describe functional and quality attributes of services, a process modeling algorithm to configure processes composed of services, a method to evaluate process utility based on quality metrics and user preferences, and an implementation that allows a user to instantiate the optimal process.

Because the framework allows a user to rapidly reconfigure and select optimal processes as new services are introduced or as requirements change, the framework should reduce burdensome software maintenance costs associated with traditional automation methods and provide a more flexible approach.

ACKNOWLEDGMENTS

The completion of this work would not have been possible without the support of many people. I am especially grateful for the experience of working with Brent Staubach and Jeff Perlak in the Systems Optimization group at Pratt & Whitney and for their role in securing funding for the project. A special thanks is also due to my committee—Dr. Jordan Cox, Dr. Jeffrey Humpherys, and Dr. Christopher Mattson—for their insight and guidance. Finally, I am thankful for the unwavering faith and patience of my dear wife Jacqueline.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 Introduction

Market globalization and mass customization requirements are forcing companies towards automation of their product development processes. The benefits of automation include reduced cost and decreased cycle time for new product introduction. Additionally, automation can provide a foundation for standardizing the product development process.

Many task-specific software solutions provide localized automation. Coordinating these local solutions to automate higher-level processes presents a significant challenge due to the incompatibility of the software tools. An even greater challenge to automation is the dynamic nature of the product development environment. Markets, competitors, technologies, and methods are constantly changing and demand flexibility in the automation approach.

Current automation methods do not provide the required level of flexibility to operate in this dynamic environment. Even small changes in the product development environment can result in large additional software investment in order to customize the implementation to the new environment. However, such an approach is analogous to shooting at a moving target.

New formalized strategies and technologies for managing and automating product development processes are needed in order to make automation cost-effective in the long-term. Not surprisingly, similar trends to those in product development automation are also occurring across many other functional areas of business. In an effort to address some of these business concerns, there have been many recent developments in the information technology industry. Some of these developments include Service Oriented Architecture (SOA) and Web services.

Despite the benefits that these new developments offer, product development processes tend to be more complex and dynamic than many other traditional business processes and will require additional support to reap the full benefits of automation. Autonomic systems and semantic web services provide possible enabling solutions when used within an overarching product development process automation strategy. This research proposes such a strategy.

The goal of this research is to provide a flexible, standardized approach to product development process automation in order to significantly reduce the software maintenance costs associated with traditional automation methods. This will be accomplished through the following objectives:

1. Formalize an approach to decompose product development process into reusable services.
2. Define functional descriptions for the services based on a product development ontology.
3. Define nonfunctional descriptions for the services based on a Quality of Service (QoS) ontology.

4. Develop a service discovery algorithm that will utilize functional descriptions to identify services that can meet specified objectives.
5. Develop a process modeling algorithm that will dynamically configure the identified services into candidate processes that can meet the specified objectives.
6. Develop a process analysis algorithm that will utilize the nonfunctional descriptions and user preferences to evaluate the overall utility of the candidate processes and identify the optimal process.
7. Develop an automated method by which a user can instantiate the identified process for execution or further development.
8. Demonstrate the flexibility of the proposed approach with a specific product development process.

2 Background

This chapter provides an overview of relevant theory and technology necessary for the development of an autonomic approach to product development process automation. This previous work includes Service Oriented Architecture (SOA), Web services, autonomic systems, software agents, and ontologies.

2.1 Service Oriented Architecture

Service Oriented Architecture (SOA) provides a flexible approach that allows a business to quickly adapt to changing conditions and requirements. Essentially SOA is an information technology (IT) architecture that supports transformation of existing IT infrastructure into a set of linked services that can be accessed when needed over a network. These services can by efficiently combined to accomplish specific business objectives [1].

For product development process automation a SOA provides a highly reusable set of automation services. Each of these services performs a single function that can be combined with other services to build larger processes [2]. This means that when business needs change, new processes can be rapidly configured without recoding software.

A significant additional benefit is that the implementation is hidden from a user who is only aware of the functionality exposed through the service interface. This reduces software maintenance costs when changes are necessary for a particular service implementation, because these changes can be isolated. This is in stark contrast to typical product development automation methods that essentially hardwire multiple external dependencies into software through traditional coding. In this latter approach, changes in business need may require extensive overhauls to the code.

2.2 Web Services

Essentially, at a high level a Web service is a unit of work that handles a specific functional task. More technically speaking, a Web service is a software interface that describes a collection of operations that can be accessed over the network through standardized XML (eXtensible Markup Language) messaging [1].

Integrating software across multiple operating systems, programming languages, hardware, and networks into larger processes requires a nonproprietary, adaptable environment. Web services are a key component of this environment. Web service standards provide a non-proprietary framework that ensures compatibility of services within a company intranet and across the Internet. The standards provide details for Web service description, messaging, and publication [3]. Web service technologies enable platform- and programming language-independent communication between applications.

Essentially, SOA and Web services provide the structure for releasing deeply locked business functions out of legacy code and exposing those functions to the enterprise in a consistent fashion [2]. For product development process automation, this

consistent representation of service functionality increases overall system flexibility. Additionally, it reduces software maintenance costs for automation projects in much the same way that part standardization reduced product maintenance costs in the 19th century.

2.3 Commercial Product Development Process Automation Tools

Several commercial tools have been developed during the past decade in an effort to improve the effectiveness of product development automation. The most popular of these tools are FIPER (Federated Intelligent Product EnviRonment) and ModelCenter.

FIPER was originally developed as a four year project sponsored by the National Institute of Standards and Technology and is currently a product managed by Engineous Software, Inc. The main purpose for its development was to create an environment that allows an engineer to easily integrate various software applications into a single process.

FIPER provides a graphical modeling environment that allows a user without software development skills to conveniently create an engineering process using a standard library of templates for common engineering tasks. These templates are well-integrated with a variety of third-party software application such as Word, Excel, CATIA, MATLAB, ANSYS, etc. Additionally, optimization and experimental designs can be conveniently configured to drive FIPER process models.

FIPER features a web-based architecture that can be used in a client-server mode in order to share engineering process models and design activities around the globe and with business partners. Additionally, the client-server mode features distributed computing capabilities.

The software also exposes an application programmatic interface (API) that can be used to develop custom templates to interface with proprietary codes or commercial applications not covered by the standard library.

ModelCenter by Phoenix Integration is another commercial integration software that is very similar to FIPER. The primary differences between the two include that ModelCenter is not integrated with as many third-party software applications, and the API documentation is not as extensive.

Several case studies have demonstrated the effectiveness of these tools in reducing cycle time and adding flexibility to process automation [24]. The tools themselves provide no overall strategy for product development process automation. However, because of the richness of the API provided in FIPER, the current implementation of this work will demonstrate how an autonomic product development process automation strategy can make use of FIPER or a similar tool to provide a significant increase in process automation flexibility.

2.4 Autonomic Systems

Despite the benefits of SOA, Web services, and commercial tools, product development processes tend to be more complex and dynamic than many other traditional business processes and will require additional strategies to reap the full benefits of automation. Theoretically, autonomic systems enable dynamic adjustments to changing environments such as the product development environment.

Autonomic computing systems are analogous to the autonomic nervous system of the human body. Processes within the human body such as heart beat, breathing,

digestion, thermal management, etc., are crucial to survival and wellbeing. Yet, these processes are not managed through conscious effort. Instead, the system can be viewed as a self-managed system that operates autonomously in behalf of the human being. Autonomic computing systems strive to provide similar self-management.

Two key components of self-management in autonomic computing include self-configuration and self-optimization [4]. Self-configuration means that the system will automatically reconfigure to best handle environmental changes. Self-optimization means that the system continuously looks for ways to optimize itself. It will monitor its components and fine-tune its workflow to achieve system goals [5].

An autonomic approach to product development process automation provides several significant benefits. Self-configuration enables dynamic process modeling that will reduce costly maintenance tasks associated with reconfiguring an automated process when changes are needed. Additionally, these reconfigured processes may exhibit higher efficiencies because concurrent and extraneous tasks can be identified and handled automatically. Self-optimization means that optimal processes can quickly be identified when changes occur. These benefits will provide a more flexible and cost-effective strategy to product development process automation.

Web services have been identified as a key ingredient in autonomic systems. Web services provide a reusable set of services. Software agents represent another key ingredient that provides the autonomic behavior necessary to dynamically configure these services. Additionally, ontologies provide the semantics needed for agents to understand their environment and to be able to properly configure services into larger processes to

meet system objectives. Web services have been discussed previously. Agents and ontologies will be discussed below.

2.5 Software Agents

Lander explains that there is no clear definition of an agent [6]. However, a common, high level definition is that an agent is a computer system capable of independent action on behalf of a user [7]. Despite lacking a rigorous understanding of what an agent is, implementations have been quite diverse and varied, spanning multiple disciplines. Several of these implementations have been in product development.

2.5.1 Agents in Product Development

The use of agents in product development is not a new idea. Karpowitz provides a broad literature review of agent system implementations in product development [8]. This literature demonstrates that agent-based systems can be used to create more flexible approaches to product development. The most relevant of this literature is summarized below.

- Agents may be used to integrate heterogeneous, knowledge-based design tools into an adaptable system [6].
- Multi-agent systems require minimal changes to existing tools and processes [9].
- Agent teams can be used in the conceptual design phase to find optimal configurations [10].
- A design-oriented model can be used with an agent system to create an automated product development system [11].

- Agents can be used to integrate product design, manufacturing analysis, and process planning in a distributed computing environment [12].
- Agent systems can be used to integrate conceptual design and process planning to optimize product form and structure and to reduce manufacturing cost [9].
- Existing agent-based systems for product development and process planning automation include PACT, SHARE, First-Link, Next-Link, Process-Link, and DIDE [12] [13].

2.5.2 Agents and Dynamic Process Configuration

Aside from general agent applications in product development, more specific research exists in using agents to configure processes. Specifically, the following research shows that agents can discover and select Web services in order to create larger applications or processes using ontologies and system languages:

- Business Process Execution Language (BPEL) can be used to express the initial social order of a multi-agent system. This language can be extended to allow agents to compose adaptable workflows of Web services [14].
- Agents can be used to build an application by selecting Web service implementations that best match the quality criteria of the application [15].
- Ontologies and semantic Web service descriptions can be used to dynamically discover potential workflows to meet system objectives [16].
- Agents can be used to create dynamic workflow for simple design tasks [6].

The above literature is very helpful in describing key technologies and methods that are necessary to enable autonomic product development process automation. The

work shows that processes can be configured dynamically and that the best services can be selected. However, additional work is still required to formulate a standardized approach for product development. In particular, ontological frameworks must be explored to describe product development service functionality and quality in order to enable dynamic discovery and selection of services for a real industry context. Next, a methodology for evaluating and optimizing larger processes composed of these services is needed.

2.6 Ontologies

Ontologies are essentially sets of terms that are organized and described in such a way that a machine can process them. For example, a weather ontology could be defined so that a software agent could give recommendations to users about what clothes they should wear. Obviously, the agent would need to know something about "rain," "temperature," "wind," etc. These sets of terms along with other important information such as attributes and relationships could comprise an ontology.

Ontologies are crucial to autonomic process automation. An ontology that describes service functionality enables the system to dynamically discover services that can meet system objectives and build larger processes using these services. An ontology to describe service quality allows the system to determine which process is best, given user quality preferences. Ontological frameworks for the Internet have been in development during the past decade and provide valuable insight into the use of ontologies for autonomic process automation within an enterprise. These frameworks can be broadly classified under the umbrella of the Semantic Web.

2.6.1 Semantic Web

The Semantic Web provides machine interpretable meaning to the current Web in a ". . . common framework that allows data to be shared and reused across application, enterprise, and community boundaries" [17]. Currently on the Web, data that is hidden away in HTML (HyperText Markup Language) files is useful in some context but not in others. On a global scale it is difficult to process this information in an efficient way. The Semantic Web can be thought of as a globally linked database that would allow users to efficiently process Web content and use it in dynamic ways [18].

For example, a traveler may be planning to fly home to attend a baseball game. This activity requires information regarding local sport events, weather, and flight schedules. Each piece of information is presented in various websites, and theoretically, it should be possible to glean necessary information from the different sites and create a travel plan. However, all the information is described using HTML, meaning that the traveler must use search engines and trial and error to manually gather all of the necessary information. In the Semantic Web a travel plan could be more dynamically generated because the diverse information could be automatically discovered and linked together.

The hype around the Semantic Web began around 1999 and has been growing steadily ever since. Despite the hype Semantic Web technologies are still in their infancies, and despite the apparent potential, there is little consensus about the likely direction and characteristics of the early Semantic Web [18].

Two key technologies for the Semantic Web are the Resource Description Framework (RDF) and the Web Ontology Language (OWL). RDF is used to represent

information and to exchange knowledge in the Web. OWL is used to publish and share ontologies, supporting advanced Web search, software agents, and knowledge management [19]. OWL is intended to provide a language that can be used to describe the classes and relations between them that are inherent in Web documents and applications [19]. An essential extension within OWL is the Web Ontology Language for Services (OWL-S) which provides support specific for Web services. OWL-S is the Semantic Web technology most relevant to this work and will be described in the next section.

2.6.2 Web Ontology Language for Services (OWL-S)

OWL-S is an OWL ontology for describing Web services. It enables users and software agents to automatically discover, invoke, compose, and monitor Web services. OWL-S consists of a set of markup language constructs. Figure 2.1 describes the three key files used in OWL-S. These files—profile, grounding and model—describe what a service does, how to access it, and how it works.

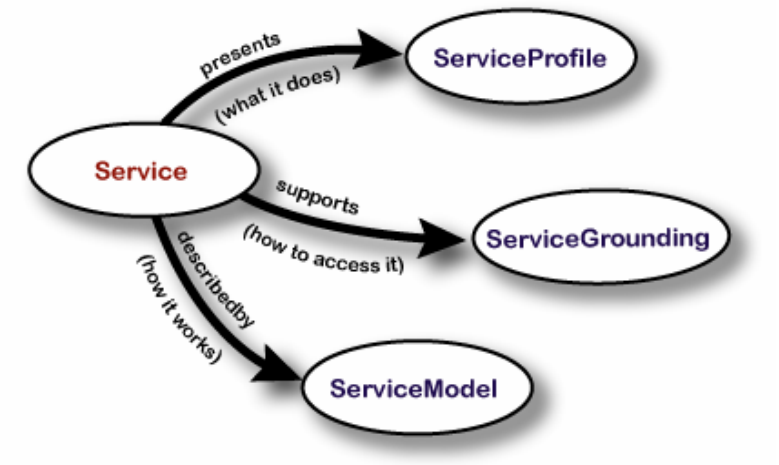

Figure 2.1 OWL-S description [21]

The traditional approach to describing a Web service is through the Web Service Description Language (WSDL). A WSDL document for a service describes inputs, outputs, and methods as defined in the service interface. This information can be made available to potential consumers of the service through a public registry with a Unique Resource Identifier (URI) to the service's WSDL. However, for a person or machine to understand the functionality of the service based solely on WSDL would require a great amount of familiarity with the service and would not allow for the service to be more dynamically discovered or used within a larger framework with ease.

With this in mind OWL-S provides an ontological representation of service functionality that goes beyond WSDL. Within OWL-S a service profile provides information regarding service inputs and outputs. Additionally, OWL-S distinguishes between basic inputs/outputs (I/O) and conditions known as preconditions and postconditions. Preconditions reflect what is required before a service can execute and postconditions specify what will be accomplished by the service.

The difference between standard I/O and these conditions is at least two-fold—conditions are normally at a higher level of abstraction and conditions may not be reflected in an actual flow of data to or from the service. In essence these conditions capture business logic. Additionally, the meaning of these conditions can be explained in a machine-readable way through the Web Ontology Language (OWL).

Essentially, WSDL does describe how to access and use a service but says nothing about what it does. OWL-S, on-the-other-hand, facilitates the automation of Web service tasks including automated Web service discovery, execution, interoperation, composition and execution monitoring [20].

The OWL-S service profile document is the key ingredient needed for dynamic discovery and selection of a Web service. The service profile is a description that describes what a service does. A diagram describing the OWL-S Service Profile can be seen below.

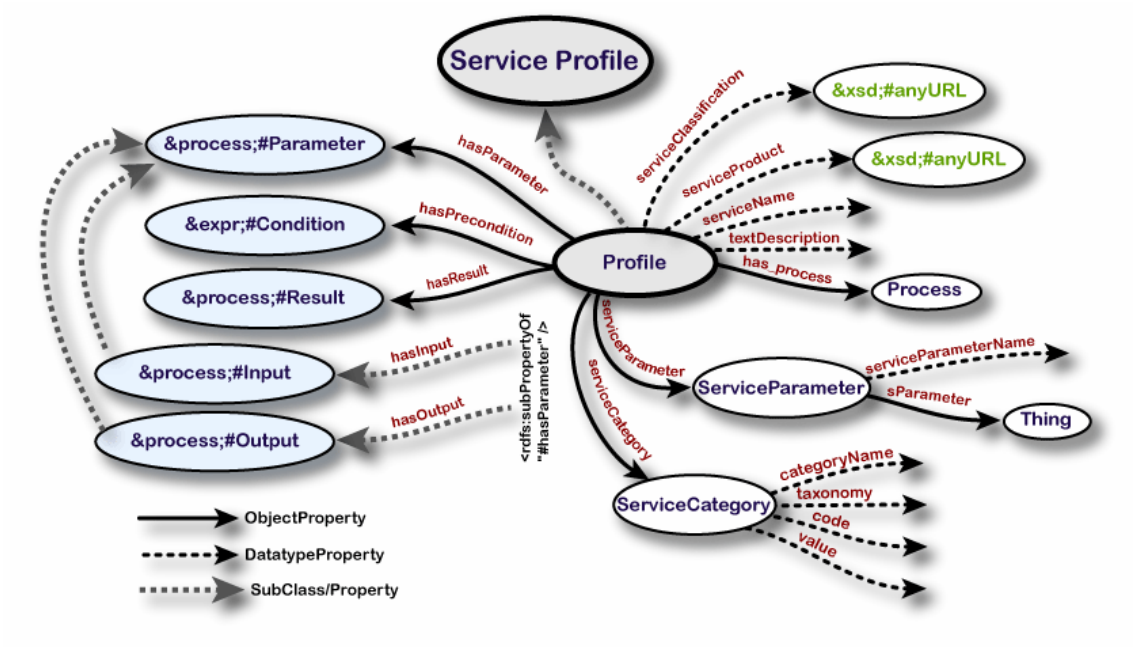

Figure 2.2 OWL-S service profile classes and properties [21]

The service profile tells what the service does, in a way that a service-seeking agent can determine whether the service meets the required objectives. The profile includes a description of what is accomplished by the service, limitations on service applicability and quality of service, and requirements that the service requester must satisfy to use the service successfully [21].

The service profile does not mandate a specific representation of services. Instead OWL subclassing can be used to create specialized service profiles. OWL-S provides

one possible representation through the class Profile. An OWL-S Profile describes a service as a function of three basic types of information: what organization provides the service, what function the service computes, and a host of features that specify characteristics of the service [21]. An outline of this information follows.

1.1. serviceName

1.2. textDescription – briefly what the service offers, what it requires, etc.

1.3. contactInformation – humans or individuals responsible for the service.

1.4. Functional Description

1.4.1. Information Transformation

1.4.1.1.Inputs

1.4.1.2.Outputs

1.4.2. State Change

1.4.2.1.Preconditions

1.4.2.2.Postconditions

1.5. Profile Attributes

1.5.1. serviceParameter

1.5.1.1.serviceParameterName

1.5.1.2.sParameter

1.5.2. serviceCategory

1.5.2.1.categoryName

1.5.2.2.taxonomy

1.5.2.3.value

1.5.2.4.code

1.6. Service Type and Product

1.6.1. serviceClassification

1.6.2. serviceProduce

For this work, the most important elements of the OWL-S Service Profile are the functional descriptions of service I/O and profile attributes. The combination of these two elements should allow a software agent to identify a compatible service (one whose preconditions and postconditions match user objectives) and the best service (a compatible service with the best quality attributes).

OWL-S is very specific in the functional description of the service which defines the information transformation (inputs and outputs) and the state changes (preconditions and postconditions). However, conceptually OWL-S provides no direction on how profile attributes should be used to represent the quality of the service. Therefore, an additional ontology extension is needed to dictate how services can be rated against one another. A Quality of Service (QoS) ontology provides this needed extension.

2.6.3 Quality of Service Ontology

QoS (Quality of Service) represents the performance properties of a service. These properties could include delay, throughput, cost, or any number of metrics. A QoS ontology provides a defined set of quality metrics for describing services that can be used by a machine to reason about which service is best under given conditions

An effective QoS ontology can be structured in order to effectively manage QoS attributes and ensure flexibility and reuse for creating domain-specific lower ontologies. Maximilien et al [26] describe a QoS ontology with three levels of abstraction. In their work the upper ontology provides generic quality definitions and relationships. The

middle ontology provides specific attributes that are common across all services. Finally, this middle ontology can be complemented by a domain-specific lower ontology. Specific, middle ontology quality attributes include availability, capacity, economic, interoperability, performance, reliability, robustness, scalability, security, integrity, and stability.

The middle ontology definition draws on substantial previous work. For example, additional suggestions for Web service quality attributes include exception handling, accuracy, accessibility, regulatory, supported standard, and completeness [28] [29]. Additionally, Chatterjee provides additional classifications for precision and accuracy for general resources within distributed systems and is not limited to Web services [30].

The QoS ontology described by Maximilien et al is generic enough to describe quality attributes for product development services. The ontology can then be complemented by additional domain ontologies within product development one at a time or in a hierarchal fashion depending on organizational needs.

3 Method

A formal method will be developed to provide a step by step procedure for implementing autonomic product development process automation. This method provides a flexible, standardized approach to product development process automation in order to significantly reduce the software maintenance costs associated with traditional automation methods.

3.1 Decomposing Product Development Processes into Reusable Services

Before product development processes can be decomposed into services, they must first be formally described. To this end, approaches such as the Product Transformation Schematic (PTS) proposed by Roach et al [22] or other graph-based methods can be used. For example, using the PTS methodology product development processes can be described mathematically in terms of transformation functions and sets of inputs and outputs that as a whole represent the design of a particular product. These transformation functions can be used as the basis for the identification of potential services as demonstrated by Karpowitz [8].

Once potential services are identified by using a process decomposition technique such as the PTS, it is still necessary to determine the level of abstraction that will be appropriate for the services and the scope of each service. Additional guidelines can be

applied to this end. For example, Young [23] recommends that services can be created in a way to maximize key attributes. These attributes ensure service reuse and other benefits of a Service Oriented Architecture. Each of these attributes is described briefly below.

1. Modular Composability – A service should be sufficiently independent so that it can be reused in new contexts.
2. Modular Understandability – A service should support one distinct concept so that it can easily be understood by a person or machine.
3. Modular Continuity – Service interfaces should hide implementation details in a way that small changes in the service do not require changes in other services or consumers of the service.
4. Direct Mapping – A service should map to a specific problem domain so that they are self-contained and independent.
5. Information Hiding – A service should not expose internal data structures.
6. Loose Coupling – A service should have few known dependencies with other services.

It is significant to understand the difference between the scope of a software application and the scope of the services which are developed that make use of the application. A software application usually provides a wide range of functionality. However, consistent with modular understandability and the other key attributes, services that wrap a particular software application should, in general, each support one unique concept and not the entire functionality of the application. This is particularly important in product development process automation in which individual services will need to be

composed into a larger business process. Following the guidelines described above will facilitate a rapid integration of these services into a larger process.

Using the discussed methodologies, the product development process can be effectively decomposed into reusable services. These services by themselves provide a solid foundation for automation, and provide needed reuse in order to reduce costly and time-intensive software coding. However, additional techniques can be applied in order to facilitate dynamic orchestration of these services into larger processes that can meet system level objectives. These techniques will be discussed in the next sections.

3.2 Defining Functional Descriptions for the Services

Functional service descriptions enable dynamic orchestration of services by ensuring that each service is used in the proper context and mapped correctly with other services. The functionality of a given service is represented by its accompanying set of inputs, outputs, and methods as defined in its interface and an accompanying WSDL (Web Service Description Language) document. However, as described previously OWL-S can be used to provide more semantic meaning to the functional description through preconditions and postconditions linked to an ontology.

The use of this higher level ontological description of service functionality in an industry context is significant. Particularly in a traditional product development setting, the lower-level I/O developed for software tool automation can be extensive and quite diverse. Likely, there will be a lack of standards to the documentation, much less particular naming conventions for parameters and methods. Even if naming conventions and standards do exist, managing this across departments, organizations, and/or business

partners is exceptionally daunting. However, by using concepts from OWL-S and related frameworks the I/O can be effectively managed at the business logic level.

This is especially advantageous for engineering analysis applications because the business logic I/O level can be abstracted into a physics-based ontology that will be particularly unaffected by typical product development environment changes related to changing markets, new products, and the latest technologies.

One example of a physics-based ontology for engineering analysis classifies analysis parameters according to an n-dimensional vector. The first element of the vector defines the discipline such as structural, thermal, fluids, etc. The second element defines the discipline-specific analysis type. The third element defines the dimensionality of the analysis. Finally, any number of additional elements can be used to provide further description within each discipline as appropriate. Each vector represents a subspace that can be populated with appropriate analysis parameters that are then uniquely defined.

Then, each service description contains a list of the vectors and analysis parameters specific to that service. These parameters can also be grouped appropriately within preconditions and postconditions.

For example, imagine that a structural analysis of a simple, statically loaded beam or other structure is required. A functional representation of a service that can provide this analysis could be as follows:

<postcondition>Analysis.Structural.Static.3D.MAX_STRESS</ postcondition >

In this example “Analysis” is the ontology being used, “Structural.Static.3D” is a three dimensional vector and “MAX_STRESS” is an acceptable keyword defined for use within the “Structural.Static.3D” subspace. A precondition to the service could be information regarding the structure geometry such as

<precondition>Analysis.Geometry.Solid.3D.PART</precondition>

It is significant to note that while the parameters “MAX_STRESS” and “PART” in this case provide a high level representation of actual dataflow to and from the service, this will not always be the case. For example, consider a billing service that has as a precondition parameter “CREDIT_CARD_EXISTS.” This parameter would be useful to describe a control flow (sequence of execution) relationship with another service that provides credit card information and has “CREDIT_CARD_EXISTS” as a postcondition. In both the structural analysis service and the billing service, the preconditions are useful to infer relationships with other services at a business logic level, and can be used to determine sequential control flow and parallel execution paths.

In the structural analysis case, the precondition also provides a generic representation of eventual dataflow but this does not have to be the case and the parameters should be thought of as abstract representations of I/O. Non-abstract parameters can be specified in the preconditions and postconditions. However, this is to be discouraged because it can enforce a more rigid coupling with other services and inhibits more dynamic interaction.

The proposed ontological representation is extendable to capture the physics-based functionality of any engineering analysis. A representation similar to the analysis vector has been used by Bailey et al [24] in their "simulation engine" to describe simulation tools for analyzing generic products. The proposed engineering analysis ontology described herein extends this framework to describe specific functional parameters within each service. For a fuller definition of the ontology see Appendix A.

Once these functional descriptions have been defined for services, it is then theoretically possible to orchestrate these services into higher level processes based on requirements for specific results. However, in a typical product development environment there may be several or many process alternatives. The best process will depend on the QoS (Quality of Service) required under the circumstances and the QoS provided by the services. Functional descriptions do not provide QoS information; therefore, service descriptions must include additional nonfunctional QoS descriptions in order to facilitate process selection among competing processes. These nonfunctional descriptions are described in the next section.

3.3 Defining Nonfunctional Descriptions for the Services

In order to analyze the overall utility of a process, it is first necessary to be able to identify quality performance of process components. A Service Oriented Architecture enables a standards based approach in which process components are embodied as services with formal descriptions. Nonfunctional QoS attributes can be defined within the service descriptions.

The QoS attributes should be based on metrics relevant to the particular organizational domain. An effective QoS ontology can be used in order to effectively manage QoS attributes and ensure flexibility and reuse for creating these domain-specific lower ontologies. Maximilien et al [26] describe a QoS ontology with three levels of abstraction that will be used here. In their work the upper ontology provides generic quality definitions and relationship. The middle ontology provides specific attributes that are common across all services. Finally, this middle ontology can be complemented by a domain-specific lower ontology. The quality attributes of the middle ontology draw on previous work in web service semantics and distributed systems [28] [29] [30].

The QoS ontology described by Maximilien et al is generic enough to describe quality attributes for product development services. The ontology can then be complemented by additional domain ontologies within product development one at a time or in a hierarchal fashion depending on organizational needs. This is important because an organization wishing to implement the proposed method may not have empirical data to support a wide variety of QoS metrics. By supporting a few key metrics initially based on available data and possibly subjective criteria, initial QoS description can be made for services that can then be adjusted later through a more elaborate tracking system.

For example, consider the engineering analysis ontology described in the previous section. Such an ontology could include initial lower ontology QoS metrics such as variable cost (subclass of cost), precision, reliability, and execution time (subclass of performance). For a particular organization, precision may be more easily measured for engineering analysis by an additional QoS metric subclass called r-squared. R-squared could represent the statistical r-squared value obtained when comparing experimental and

historical data. The ontology would also likely include information such as units and limits. An example of such an engineering analysis ontology can be found in Appendix A and will be used later to demonstrate a specific implementation.

3.4 Service Discovery and Process Modeling Algorithm

Implementations of the procedures described in this section will depend on the specific software tools being used. An implementation will be described later using FIPER. However, the concepts are presented at a high level in this section in order to delineate between theory and practice.

Once services have been created and described, they are ready to be published and registered with the system. Registration includes updating a system registry including a functional map ("yellow pages") and service provider map ("white pages"). The functional map consists in an alphabetical listing of functions (postconditions). The entry for each function includes all potential registered providers. In the provider map, on the other hand, services are listed in alphabetical order and a list of functions and nonfunctional attributes are listed under each service entry. By maintaining these two sets of registry maps, queries can be performed efficiently.

After the services are registered a user can query the system for one or more postconditions. The system then searches the functional map and identifies all services that provide the specified postcondition. Then the system instantiates each of the candidate services and places each of them into a new process model. Within each process model, each of the preconditions of the newly instantiated service becomes a new query to the system. If the new search returns more than one candidate process, then the

process model is cloned for each additional candidate. A graphical description of the algorithm is shown in the figure below.

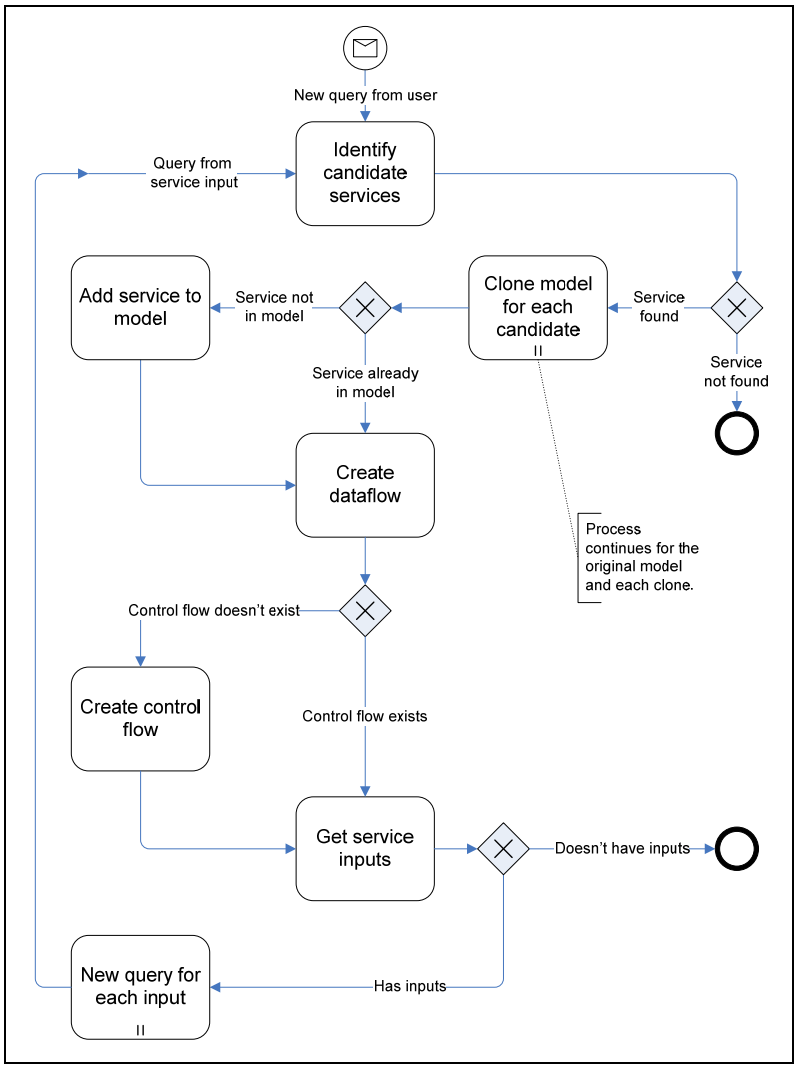

Figure 3.1 Process modeling algorithm (Business Process Modeling Notation [34])

As the algorithm proceeds, dataflow connections are made between matching postconditions and preconditions. Usually these dataflow connections are between abstract parameters but can also be between non-abstract parameters. In the case that they are abstract, the main purpose is to provide a visual connection between services that

is viewable by a human for later use. Additionally, this provides an abstract process definition that can then be implemented in different ways depending on product data. In the case that the parameters are not abstract, the process may be immediately executable, and the process modeling algorithm is effectively determining the most efficient routing within the implemented automation.

The algorithm continues this backwards mapping approach until no more candidate processes are returned for preconditions or all of the preconditions have been satisfied. Unsatisfied preconditions are identified as requiring attention by a user. A user then needs to look at the unsatisfied preconditions and decide if they can be satisfied through user input or will require creation of a new service.

Another significant feature of the algorithm is that it allows for multiple queries to be made by a user simultaneously. In this case, the algorithm must first check to see if results of already instantiated services are available in order to reduce redundancy in the process models. If an instantiated service already provides the queried result, then the algorithm simply tags that result as a match for the query and continues the search. This can be extremely useful in building a process model that requires multiple results.

The algorithm also flags all queries so that a user can quickly distinguish them from other service outputs. Execution of the algorithm yields one or more process models that can provide the queried results.

3.5 Process Analysis and Selection

Once process models have been composed, they can then be analyzed based on QoS descriptions in the services. In the dynamic product development environment

multiple functionally similar processes may exist with similar results. The best of these candidate processes will provide the best overall quality. Overall quality is not a static measure and depends on user preferences. For example, in some cases it would be desirable to select a process with the highest reliability while in another case time may be the dominating constraint. Overall quality can be represented with a utility function as follows:

$$U = \sum_{i=1}^{n} Q_i P_i \quad \textbf{(3.1)}$$

where U is the utility of a given process and Q_i and P_i are respectively the service quality and user preference for each quality attribute i. Q_i should also be normalized into z-scores as recommended by Maximilien et al [33] in order to appropriately compare different quality attributes that may use different scales.

Calculating Q_i will depend on the specific quality attribute and its corresponding aggregation function. Some specific aggregation functions have been identified in other research [27]. Aggregation functions that will be used in analyzing a later example include cost, precision, reliability, and execution time and will be described below.

The total cost of a particular process composed of services with solely variable costs can be defined as

$$C_i = \sum_{j=1}^{n} v_j t_j \quad \textbf{(3.2)}$$

where v_j and t_j are respectively the variable cost and execution time for each service *j* that belongs to a process *i* from above. This could be an appropriate cost model for an engineering analysis scenario in which software licensing agreements for particular tools could be in terms of a fixed amount of money per unit time of use.

Precision is easily calculated by finding the value of the service with the minimum precision level as follows:

$$P_i = \min\{v_j\}. \quad \textbf{(3.3)}$$

Process reliability is a product of the reliability ratings of all services in the process. Thus,

$$R_i = r_1 r_2 ... r_n. \quad \textbf{(3.4)}$$

Execution time is computed in a sequential path by summing up the execution times of each service in the path. For a more general case in which a process may contain parallel paths, the execution time is the maximum execution time found across all possible sequential paths. In equation form this gives

$$T_i = \max\{T_p\} \quad \textbf{(3.5)}$$

where

$$T_p = \sum_{k=1}^{n} t_k. \quad \textbf{(3.6)}$$

In the above, T_i is the execution time for process *i*, T_p is the sum of the execution times for each possible path *p*, and t_k is the execution time for each service *k* in each path *p*.

These aggregate functions are summarized in the table below.

Table 3.1 QoS aggregation functions

Criteria	Aggregation Function
Cost	$C_i = \sum_{j=1}^{n} v_j t_j$
Precision	$P_i = \min\{v_j\}$
Reliability	$R_i = r_1 r_2 ... r_n$
Execution Time	$T_i = \max\{T_p\}$, $T_p = \sum_{k=1}^{n} t_k$

After applying the specific aggregate functions, the process utility can be calculated as described and the process with the highest utility is selected as the optimal process. As user preferences change, the system can readjust the utility scores accordingly and find the new optimal process.

It should be pointed out that this method for optimizing processes is a global rather than local optimization method. This is due to the fact that decisions on individual service selection based on QoS are never made. Instead process selection decisions are made based on overall process utility after services have been selected based on purely functional criteria. Zeng et al point out this distinction and provide an additional local method [27]. Although, their research focused on runtime service selection whereas this work is purely design time. This design time perspective for product development is an

extremely important preliminary to any future runtime research because it complements the current role of human designers who will want, at least initially, to have a static runtime process in order to evaluate the effectiveness of the proposed automation framework.

3.6 Process Instantiation

Once the optimal process has been identified, a user will need to instantiate the process or an alternate process in order to execute or further configure the process. This is easily enabled by using a formal process modeling language such as Business Process Execution Language (BPEL) or any other sophisticated XML-based process modeling language supported with a design time and runtime environment. FIPER makes use of a proprietary XML-based process modeling language. Because the proposed method is currently implemented through software extensions on top of FIPER it is very easy to instantiate, modify, and execute models through a graphical user interface driven by an underlying XML-based process model.

As described previously, services connected through abstract data types represent an abstract process. This abstract process can then be further configured by an engineer. For example, specific product data such as computer-aided design (CAD) files may need to be pointed to within a particular service. Also additional dataflow connections may need to be made between non-abstract parameters by a human user. These non-abstract parameters may not adhere to any particular ontology and it would therefore not be possible for the process modeling algorithm to find connections between these parameters.

Once the abstract process has been instantiated and any necessary configuration steps have been made, it can then be executed. As an alternative to execution, the entire process or parts of the process could be published into the registry to constitute a new service that would then be more sophisticated than the initial services used to create it. This new service would then be available in the registry to be used dynamically along with the other services.

4 Implementation

The proposed approach to autonomic product development process automation has been implemented through software customization on top of FIPER. As described previously, FIPER provides a sophisticated process modeling language that can be used to represent processes. The processes can be modeled through an intuitive graphical interface called the Design Gateway (see Figure 4.1).

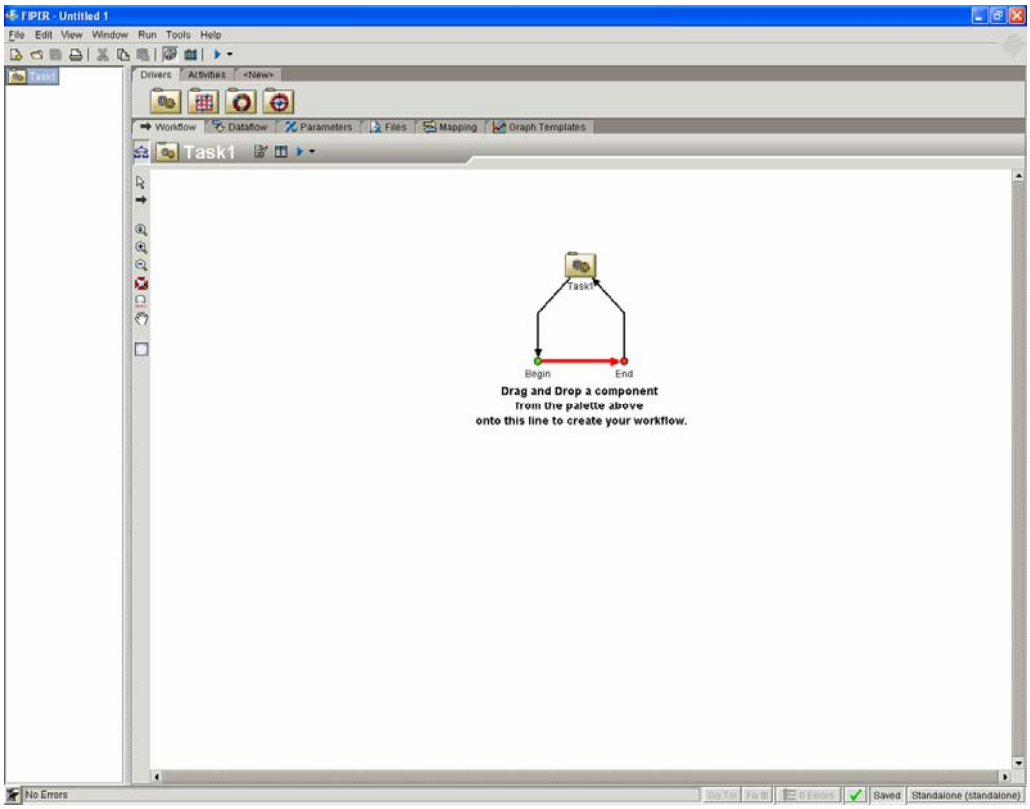

Figure 4.1 FIPER Design Gateway

The FIPER architecture is essentially web-based and supports publishing and executing services across a network. Additionally, FIPER provides an extensive application programmatic interface (API) in Java that can be used to efficiently develop services. This research also demonstrates that the API can also be used effectively as an interface with middleware in order to discover and select services, and model processes dynamically.

An example process will now be described in order to illustrate the implementation and highlight key aspects of the proposed approach. This process is an automated engineering analysis for an impeller design. A similar analysis was presented in Karpowitz [8]. This similar analysis will be used in order to highlight improvements over the previous implementation as suggested by Karpowitz as well as to demonstrate other key differences in the approach.

4.1 Service Decomposition

The first step in implementing the autonomic automation strategy is to identify the reusable services that can accomplish the individual process tasks. The previously described method for identifying services was applied. As in the Karpowitz example, the process 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.

After identifying these tasks, specific CAx tools were selected for each task and the necessary functionality of each was wrapped in a service. Four services were developed including a CATIA service, HyperMesh service, FLUENT service, and an ANSYS service. The CATIA service updates the geometric models. The HyperMesh service creates meshes based on the geometry. The FLUENT service determines surface pressure for the air solid wedge using the mesh. Finally, the ANSYS service determines the maximum stress values for the structural wedge.

Additionally, in order to introduce complexity into the example four additional services were also created—CAD2, Mesh2, Fluids2, and Stress2. As the service names imply each of these additional services is identical to a corresponding previously created service (e.g. CATIA and CAD2 are identical). This added complexity simulates an industry product development setting in which multiple tools, methods, and technologies may provide similar results. The flexible automation approach presented here will identify all potential process configurations using the available services and then identify the optimal process appropriate for a given set of requirements.

For simplicity all of the services were implemented using FIPER Script components rather than Web services. The Script component is a basic template that allows a user to insert Java code inside the component to support any desired functionality.

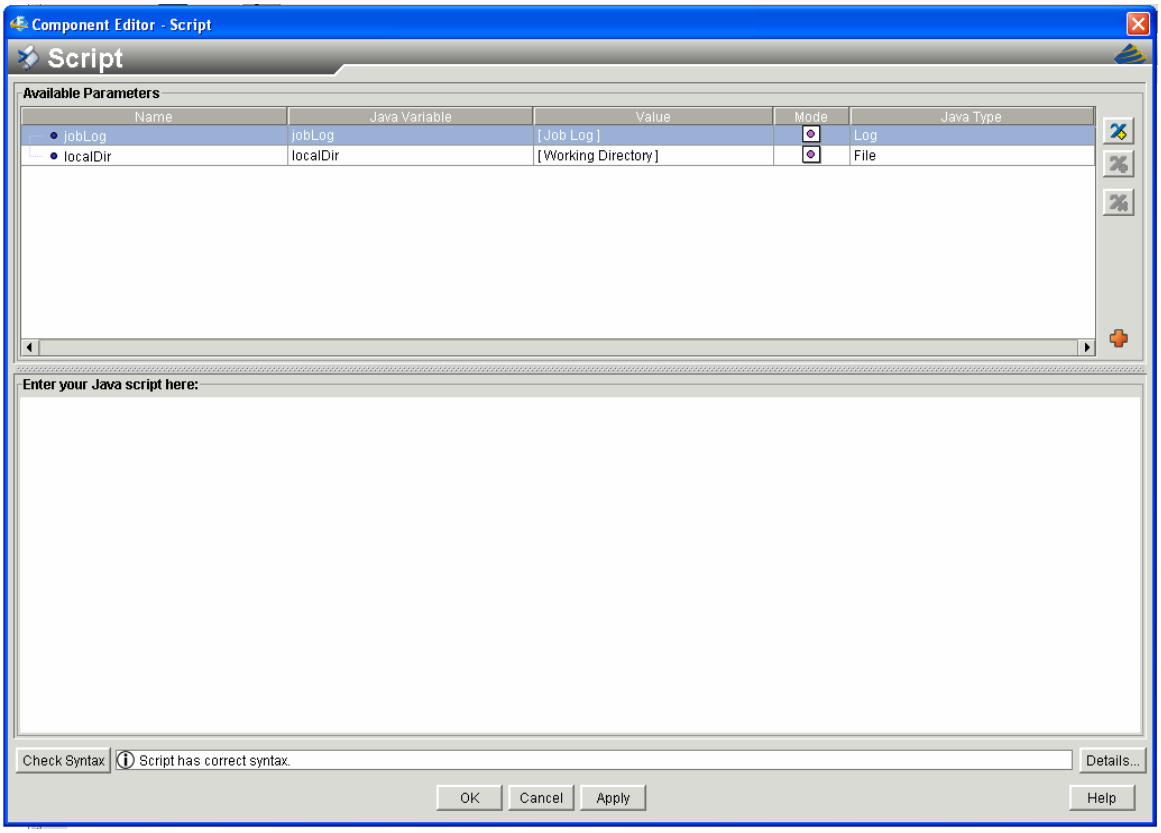

Figure 4.2 FIPER Script component

The Script uses the Dynamic Java interpreter [35] to run the Java code. Additionally, any necessary input and output parameters can easily be attached to the component to facilitate dataflow between other services. If a Web service implementation is used instead, then this service can be linked into the FIPER environment via a FIPER Web service component or through developing a custom component to wrap the Web service.

4.2 Functional Descriptions

As identified throughout this work, it is important to describe the function of the services at a business logic level so that services can be compared and matched within a

robust ontological framework. For example, although a CATIA service is currently implemented to provide a solid modeling service for the impeller, the CAD2 service also provides the same functionality. Without a scaleable strategy for describing inputs and outputs on each of these services, there would be no way to enable dynamic interactions between services to enable greater system flexibility. As already shown these higher level functional descriptions can be defined using preconditions, postconditions, and an ontology to manage naming conventions.

To implement these functional descriptions in FIPER a new data type was created. FIPER parameters can be of several types including Boolean, Real, and String. Additionally, new types can be implemented. In this case an "Abstract" type was developed. The type is called Abstract because it cannot be used to represent real data. It is used to describe preconditions and postconditions. It is a different type so that it will flag the user and the system that it is distinct from other data types. It is important to note that under a normal Web service implementation this additional functional description could be achieved by using the OWL-S (Web Ontology Language for Service). The Abstract types implemented in this work are an analogy to the OWL-S preconditions and postconditions, albeit a simplified one adapted for the FIPER environment.

The conditions or abstract parameters for the services were defined using the engineering analysis ontology previously described and are summarized in the table below.

Table 4.1 Service abstract parameters (simplified names)

Task	Abstract Input Parameters	Abstract Output Parameters	Services
Stress analysis	PART, TOTAL_PRESSURE	MAX_STRESS	ANSYS, Stress2
Fluid analysis	MESH	TOTAL_PRESSURE	FLUENT, Fluids2
Mesh creation	PART	MESH	HyperMesh, Mesh2
Geometry update	MASTER_PARAMETERS	PART	CATIA, CAD2

The parameter and service names used in Table 4.1 are simplified for convenience. The full names can be found in the full service descriptions found in Appendix A, and are consistent with the engineering analysis ontology previously described.

Abstract parameters for the CATIA service implemented in FIPER are shown in Figure 4.3.

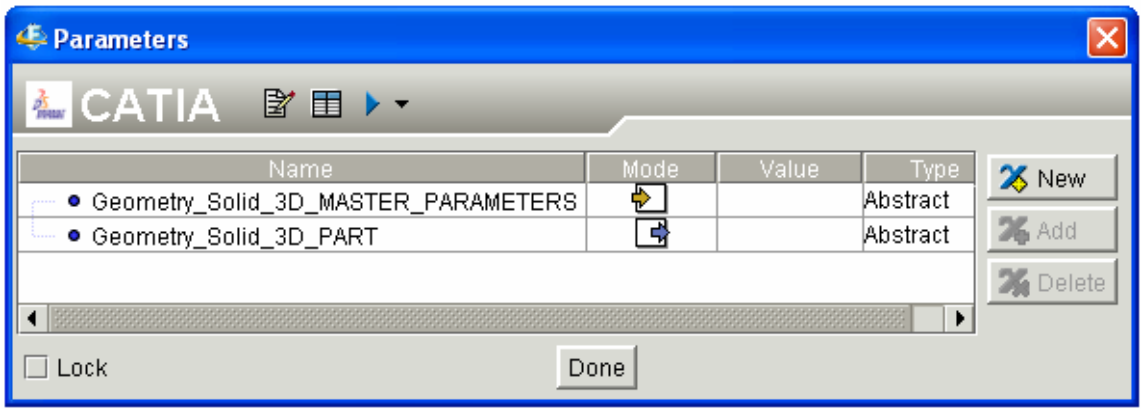

Figure 4.3 Abstract parameters

Additionally, these abstract parameters can be organized using FIPER aggregate parameters if desired (see Figure 4.4). In this way many abstract parameters could be organized efficiently according to the vector subspaces described previously for the engineering analysis ontology (e.g. “Analysis.Geometry.Solid.3D”).

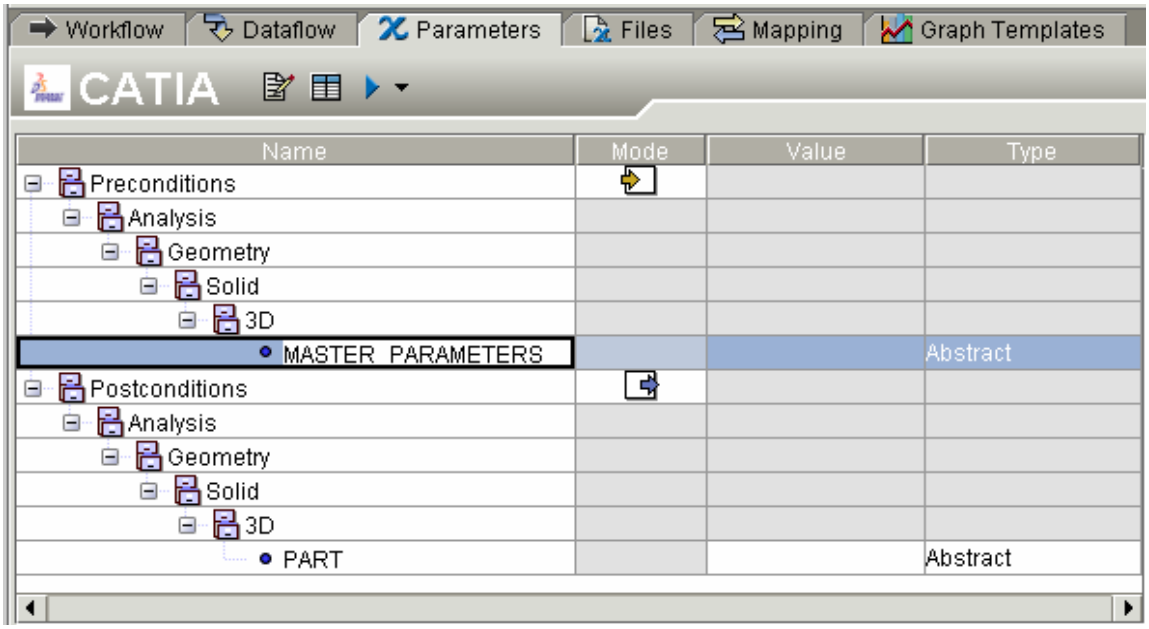

Figure 4.4 Abstract parameters organized in aggregate parameters

4.3 Nonfunctional Descriptions

QoS (Quality of Service) metrics were also defined for each service based on the engineering analysis ontology. These metrics included variable cost, execution time, reliability, and precision. These metrics are summarized in Table 4.2. A complete service description including these QoS metrics can be found in Appendix B.

Table 4.2 Quality of Service descriptions

Service	Variable Cost	Execution Time	Reliability	Precision
CATIA	11.75	19	0.80	1.00
CAD2	9.75	28	0.81	1.00
HyperMesh	1.75	30	0.90	1.00
Mesh2	0.75	27	0.85	1.00
FLUENT	1.75	90	0.95	0.87
Fluids2	2.75	50	0.97	0.94
ANSYS	3.75	20	0.95	0.91
Stress2	4.75	15	0.94	0.84

The QoS descriptions were each placed in the FIPER component description fields as shown below.

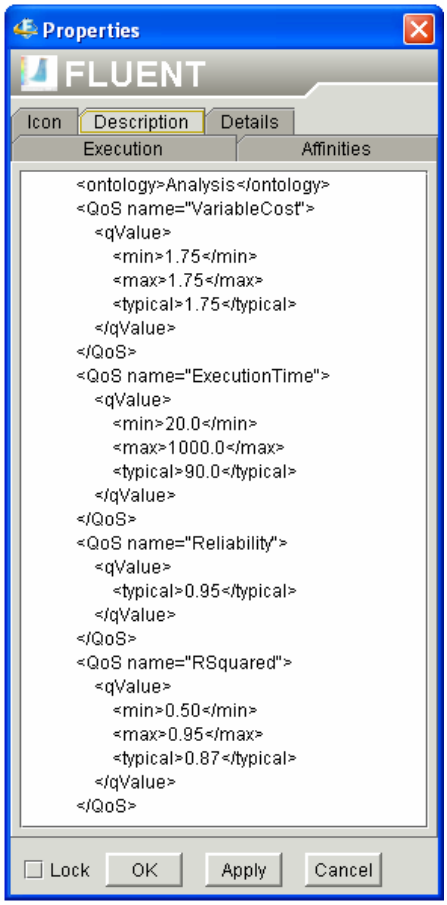

Figure 4.5 QoS description

In the absence of empirical data, values for the metrics were chosen somewhat arbitrarily except in the case of execution times for which some data existed. In future implementations these ratings could be monitored and adjusted dynamically using other agent-based frameworks described previously.

4.4 Publishing

After each service was created and described with both functional attributes (abstract parameters) and nonfunctional attributes (QoS metrics), each service was published into the FIPER library as a FIPER model (see Figure 4.6 and Figure 4.7).

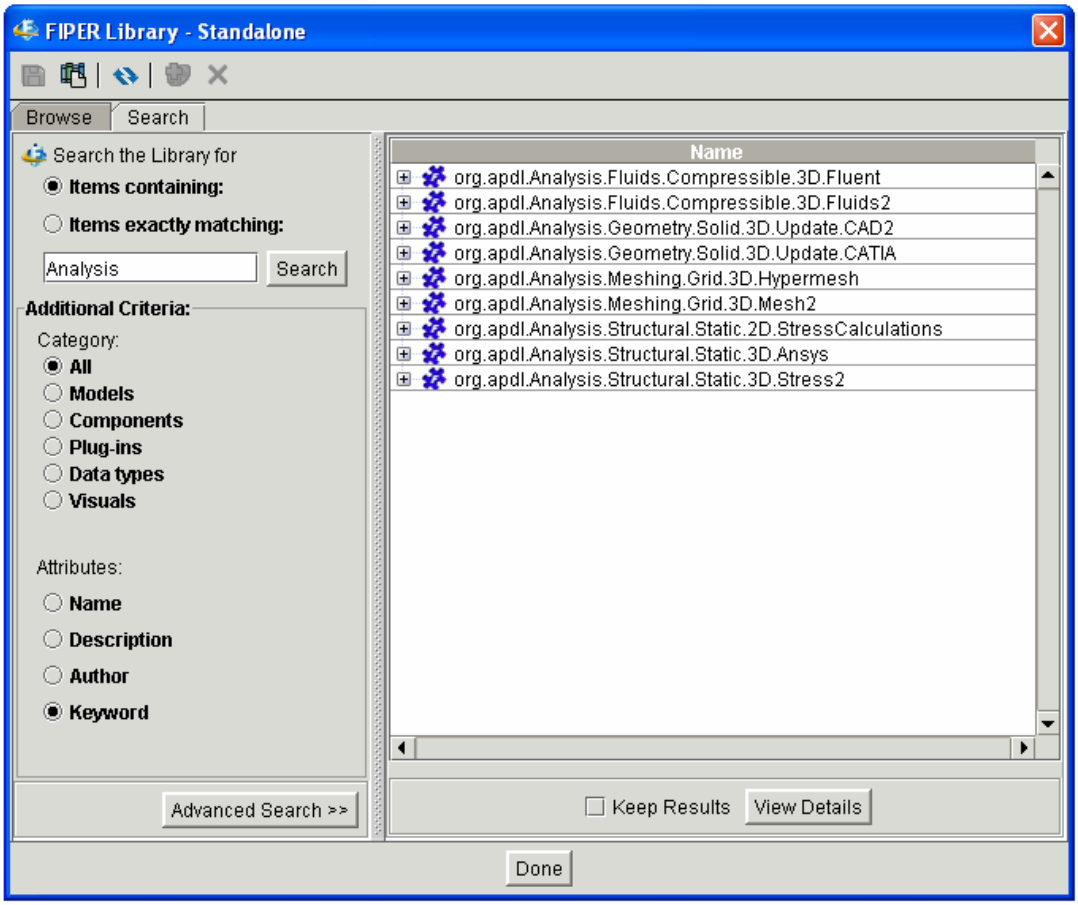

Figure 4.6 Published services listed in the FIPER library

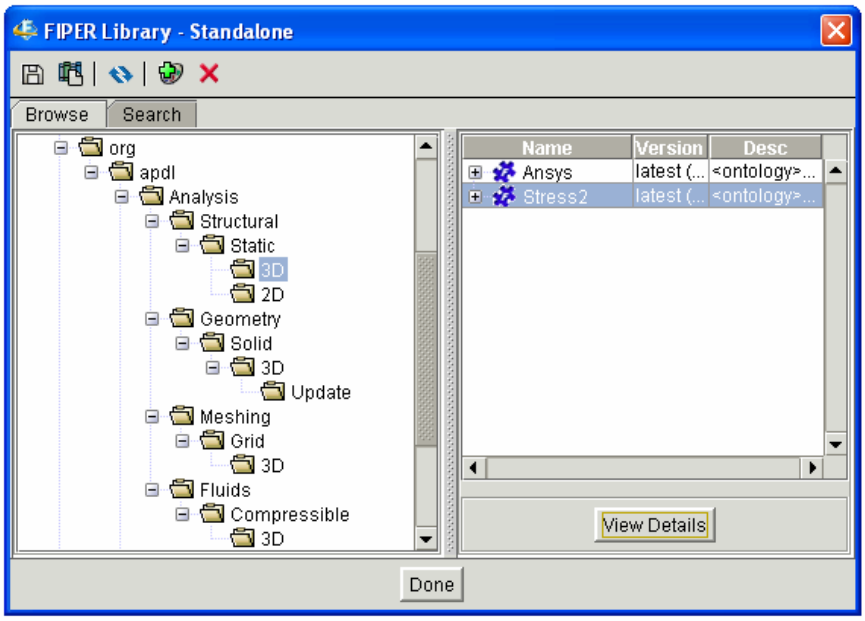

Figure 4.7 Service organization in the FIPER library

Library attributes were used to describe the QoS metrics for each service as shown below.

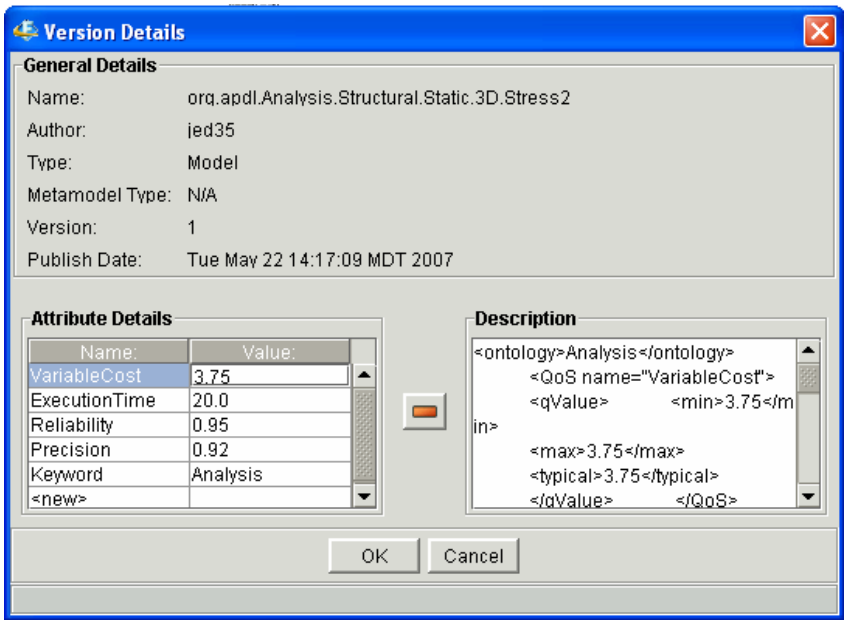

Figure 4.8 Service description

4.5 FIPER Process Manager

Once services are published into the FIPER library, they can then be combined together into larger processes. FIPER functionality allows a human user to create processes using a graphical interface with “drag and drop” functionality. Dynamic selection and configuration of services into processes by an agent or another software application is not supported in FIPER but was implemented in Java using the FIPER API.

The new application, the FIPER Process Manager, has the full functionality necessary to implement autonomic product development process automation. Key features of the Process Manager include:

- FIPER library access
- Registry management
- Service discovery and process modeling
- Process analysis
- Process optimization
- Process instantiation into the standard FIPER Design Gateway

These features will be described in more detail using the impeller design example.

4.5.1 FIPER Library Access

A view of the FIPER Process Manager can be seen in Figure 4.9. The Process Manager contains a “Controls” panel at the top and a “Views” panel at the bottom. The Controls panel is used to query the Process Manager in order to configure and optimize processes. The Views panel is used to display results returned by the Process Manager based on the queries.

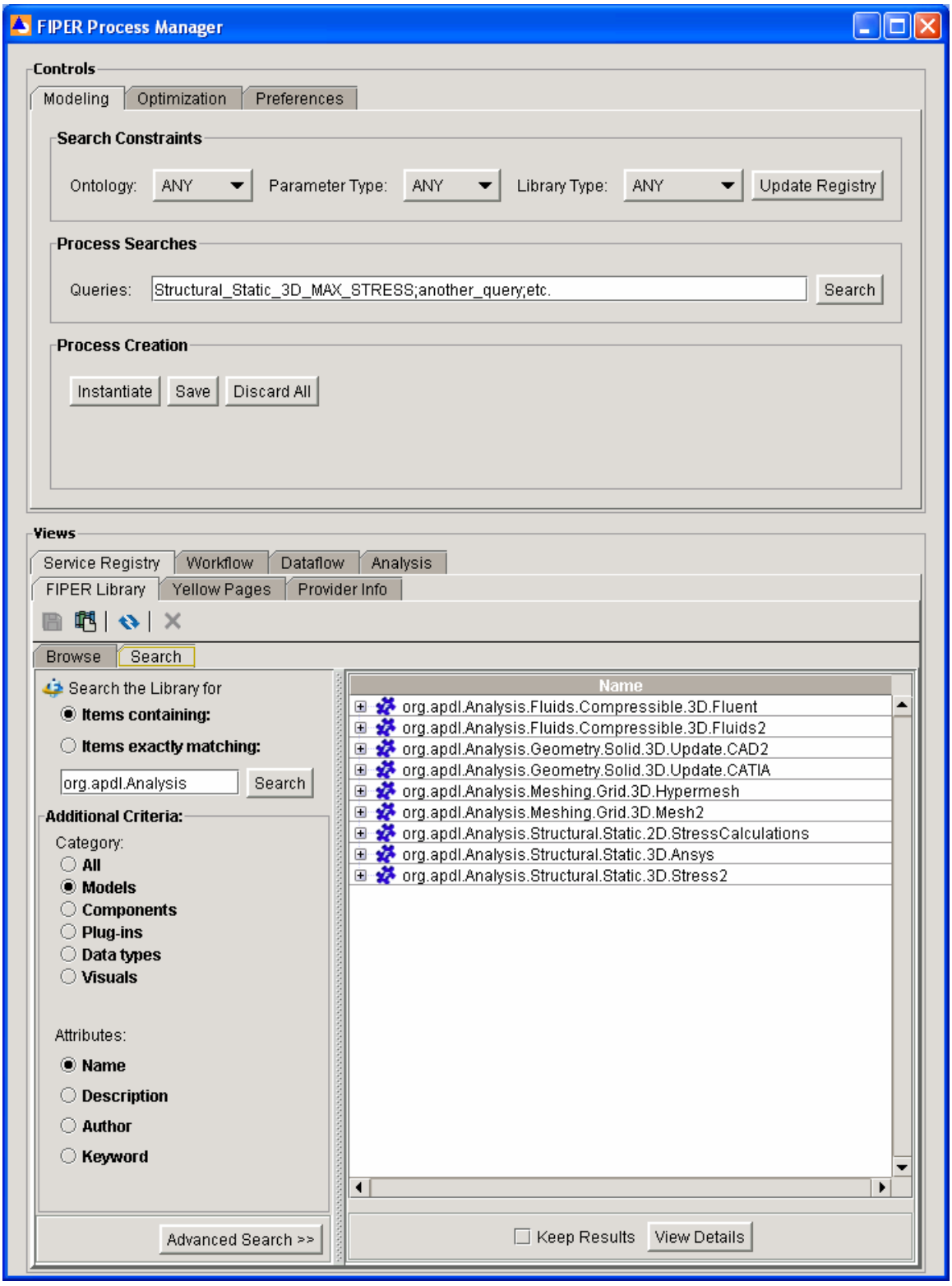

Figure 4.9 FIPER Process Manager

The Views panel contains four tabs: “Service Registry,” “Workflow,” “Dataflow,” and “Analysis.” The Workflow and Dataflow tabs display currently configured processes. The Workflow tab provides a view of the sequence of process steps while the Dataflow tab shows the flow of data or message flow between services. Finally, the Analysis tab provides analysis information based on the process QoS.

The Service Registry tab contains three tabs: “FIPER Library,” “Yellow Pages,” and “Provider Info.” The FIPER Library tab provides a standard view of the FIPER library. The Yellow Pages tab provides a list of available outputs and corresponding service providers that are published in the FIPER library. Provider Info describes each of the service providers including inputs, outputs, and quality metrics.

As shown in Figure 4.9, all of the impeller services previously described were published into the FIPER library and are viewable in the FIPER Library tab. All other FIPER library functionality is also available here.

4.5.2 Service Discovery and Process Modeling

The Controls panel contains three tabs: “Modeling,” “Optimization,” and “Preferences.” The Optimization tab is used to select the optimal process. The Preferences tab can be used to set various options such as whether to configure process paths in parallel or to configure paths only in series.

The Modeling tab contains three panels: “Search Constraints,” “Process Searches,” and “Process Creation.” Search Constraints acts as a filter to extract information from the FIPER library about available services and to populate the Yellow Pages in the Service Registry. For example, Figure 4.10 shows the Yellow Pages updated with only services of the designated ontology, parameter type, and library type.

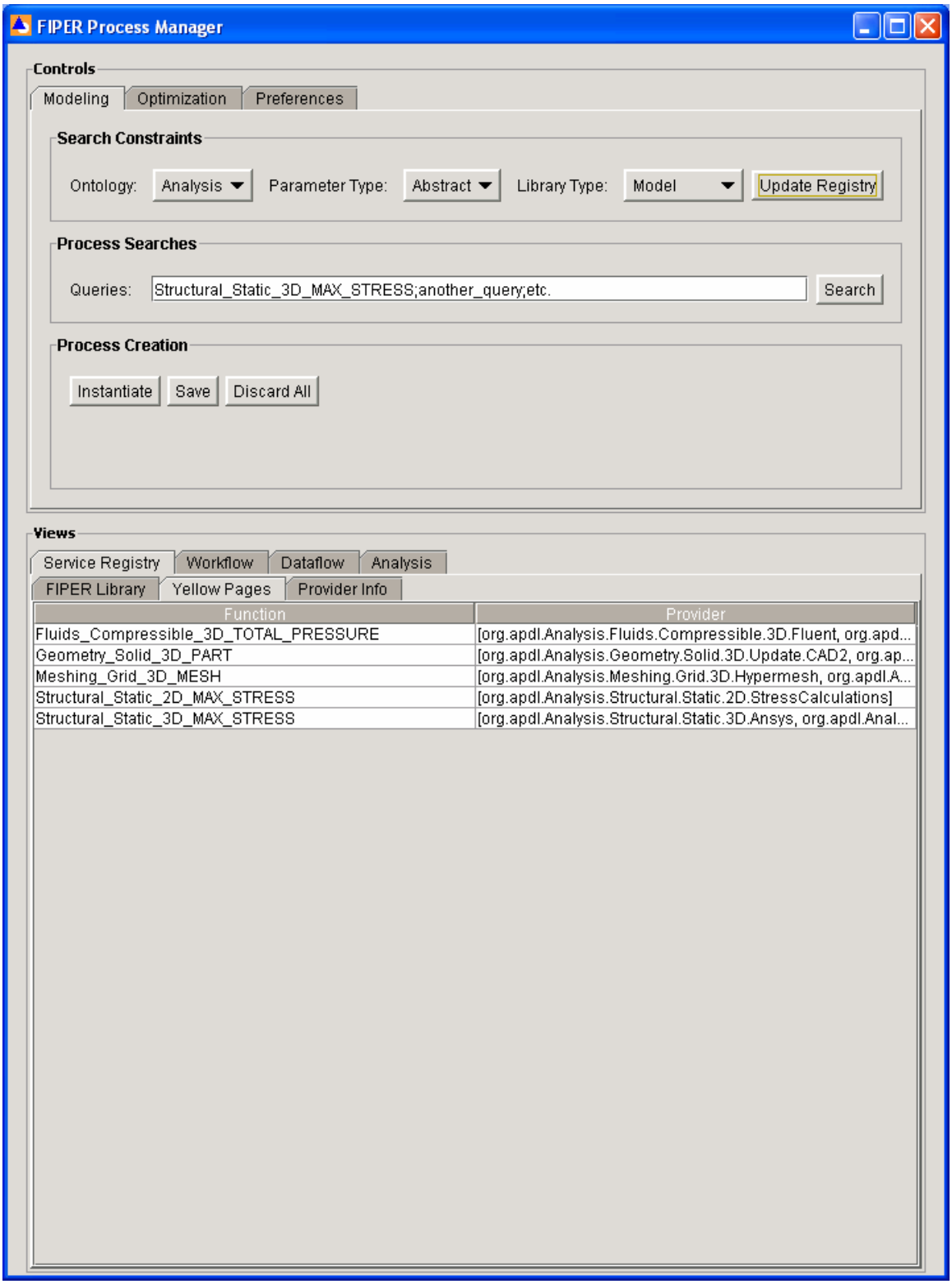

Figure 4.10 Search constraints applied to the Yellow Pages

A significant aspect of applying filters for the Yellow Pages is that this provides a way for the process modeling algorithm to more efficiently search available services to configure processes.

In the Process Searches panel queries can be submitted for new process searches. After processes have been created, they can be saved, discarded, or instantiated into the standard FIPER Design Gateway using the Process Creation panel.

For the impeller design case a query of "Structural_Static_3D_MAX_STRESS" was submitted using the Process Searches panel tab. The Process Manager then executed the process configuration algorithm described previously and created the sixteen process combinations that could provide the queried result. All sixteen processes were then displayed to the user in the Views panel as seen in Figure 4.11 through Figure 4.27. In this case the process configuration was straightforward because there are four process steps with two possible services at each step for a total of sixteen combinations.

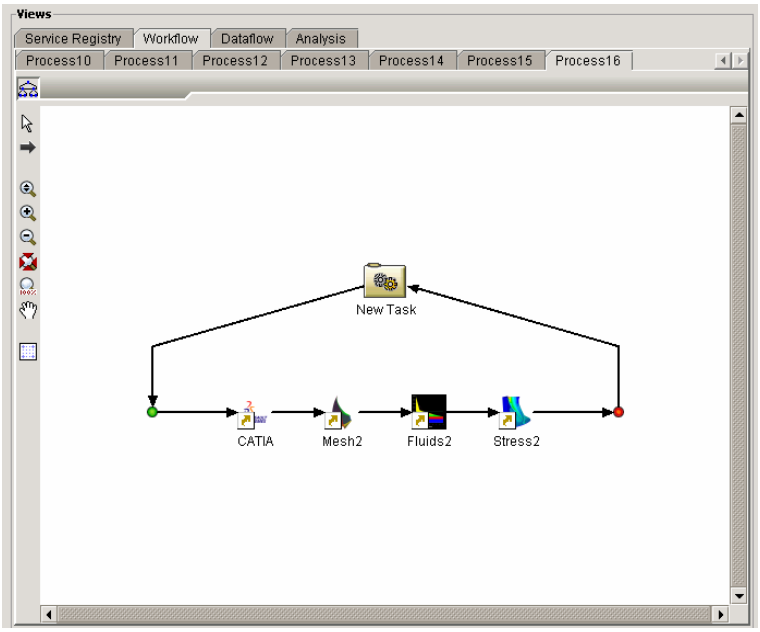

Figure 4.11 Sixteen impeller design process alternatives

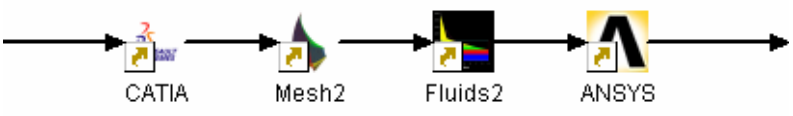

Figure 4.12 Impeller design process 1

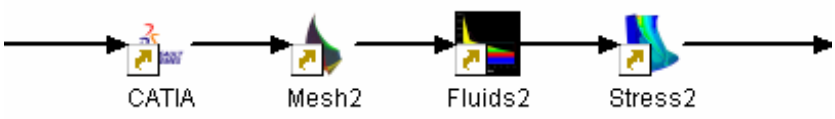

Figure 4.13 Impeller design process 2

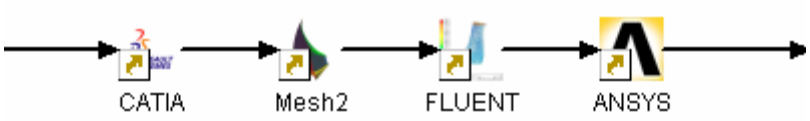

Figure 4.14 Impeller design process 3

Figure 4.15 Impeller design process 4

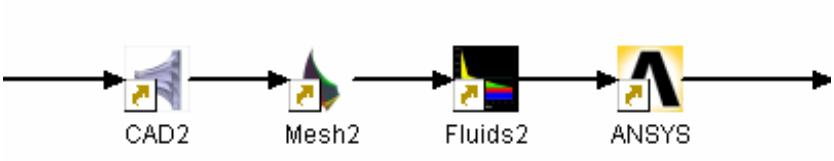

Figure 4.16 Impeller design process 5

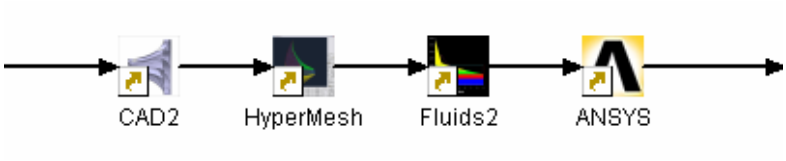

Figure 4.17 Impeller design process 6

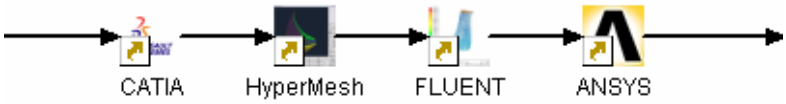

Figure 4.18 Impeller design process 7

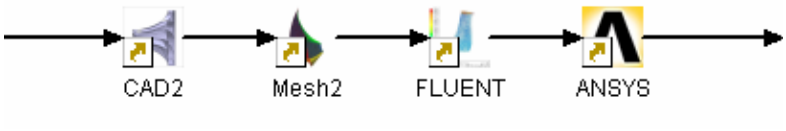

Figure 4.19 Impeller design process 8

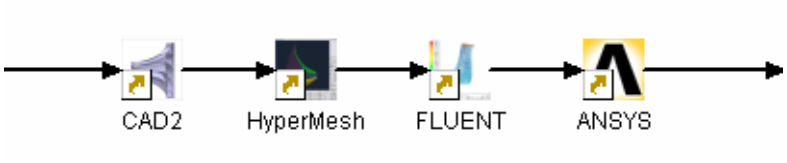

Figure 4.20 Impeller design process 9

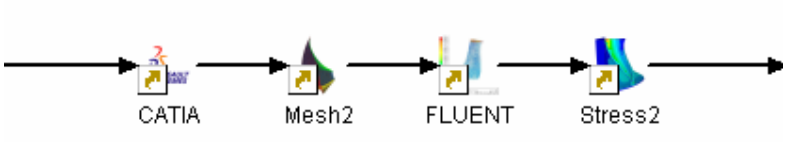

Figure 4.21 Impeller design process 10

Figure 4.22 Impeller design process 11

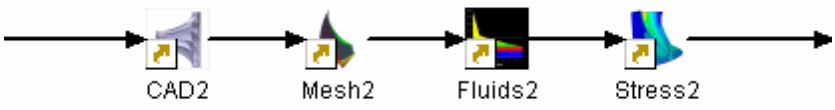

Figure 4.23 Impeller design process 12

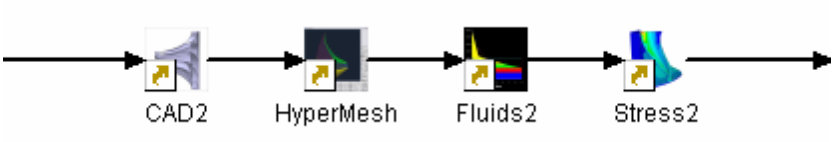

Figure 4.24 Impeller design process 13

Figure 4.25 Impeller design process 14

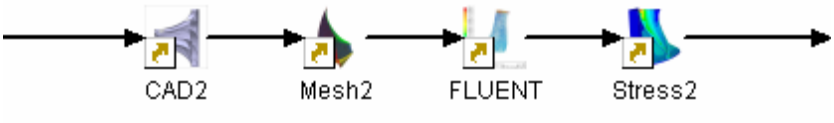

Figure 4.26 Impeller design process 15

Figure 4.27 Impeller design process 16

In addition to configuring the services in the proper sequence, the dataflow links were also created based on matching the abstract parameters (see Figure 4.28).

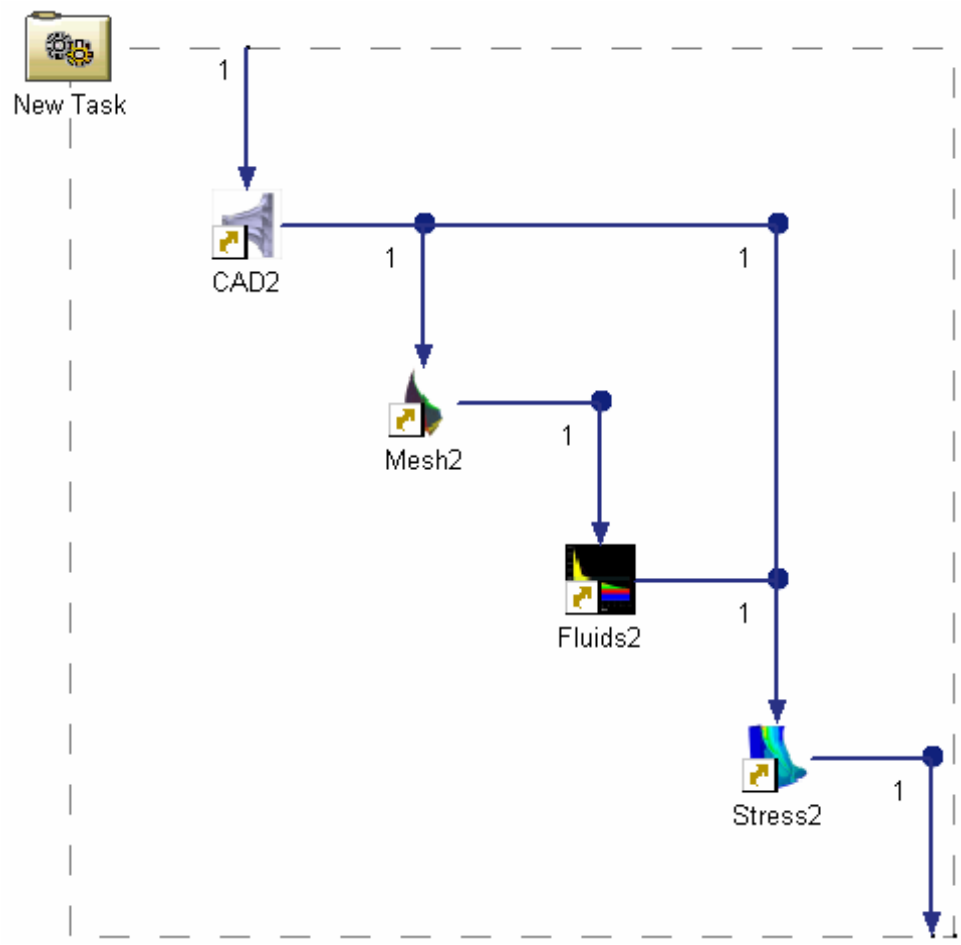

Figure 4.28 Dataflow schematic

4.5.3 Process Optimization

After the sixteen processes were configured, it was then necessary to select an optimal process. In order to perform this optimization, the aggregate QoS must be computed for each process. The aggregate QoS functions described previously (see Table 3.1) were implemented and the Process Manager applied these functions to each of

the processes and displayed this information in the “Analysis” tab (see Figure 4.29). The absolute values of the aggregate QoS values were displayed as well as normalized, z-score values.

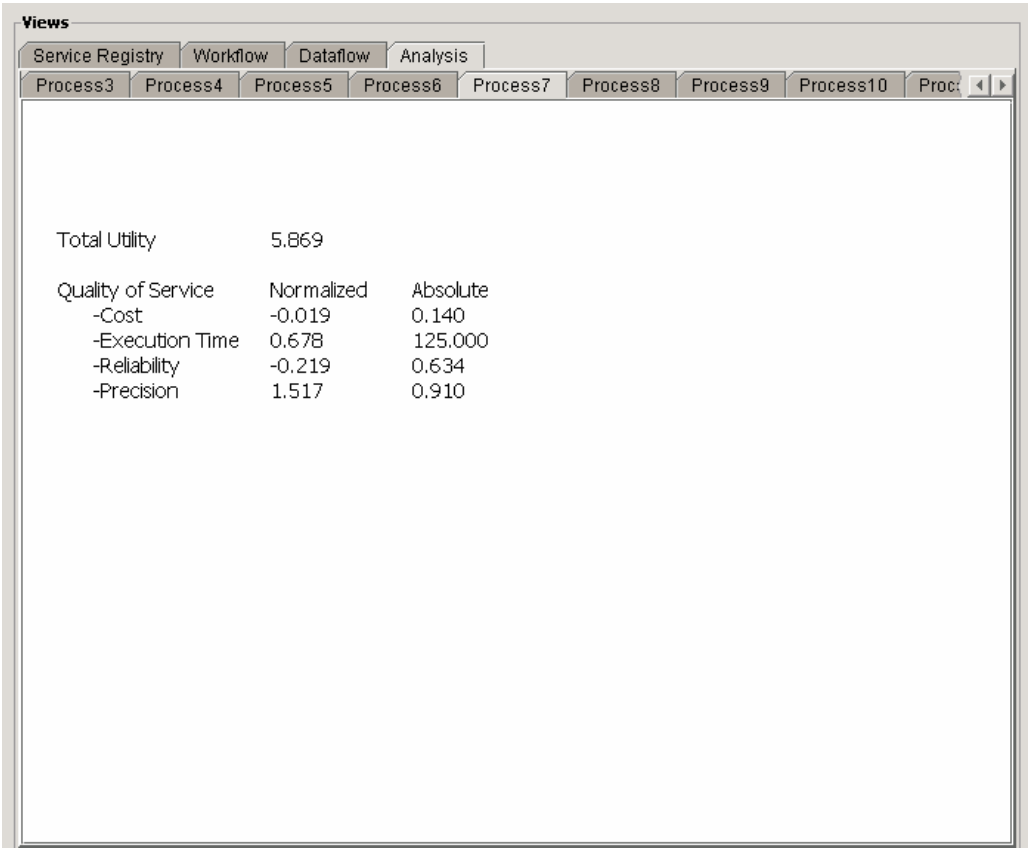

Figure 4.29 Process analysis

The total utility of each process is the sum of the products of the z-score values and the corresponding user preferences. These total utilities are additionally displayed conveniently in the “Optimization” tab along with a panel to specify user QoS preferences (see Figure 4.30). In this tab the user can specify new preferences and re-optimize. To re-optimize, the Process Manager applies the new QoS preferences to calculate new process utilities. The process with the highest utility is then identified as

the optimal process. As seen in Figure 4.30 through Figure 4.32, the optimal process can change based on changes in QoS preferences.

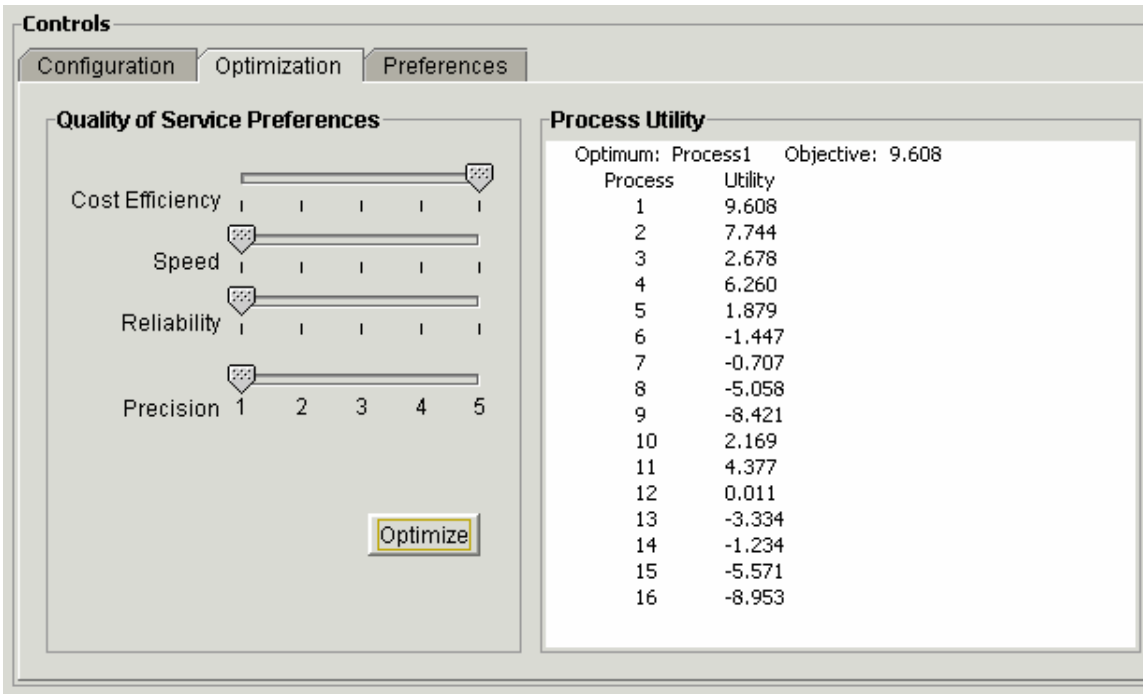

Figure 4.30 Optimizing for cost

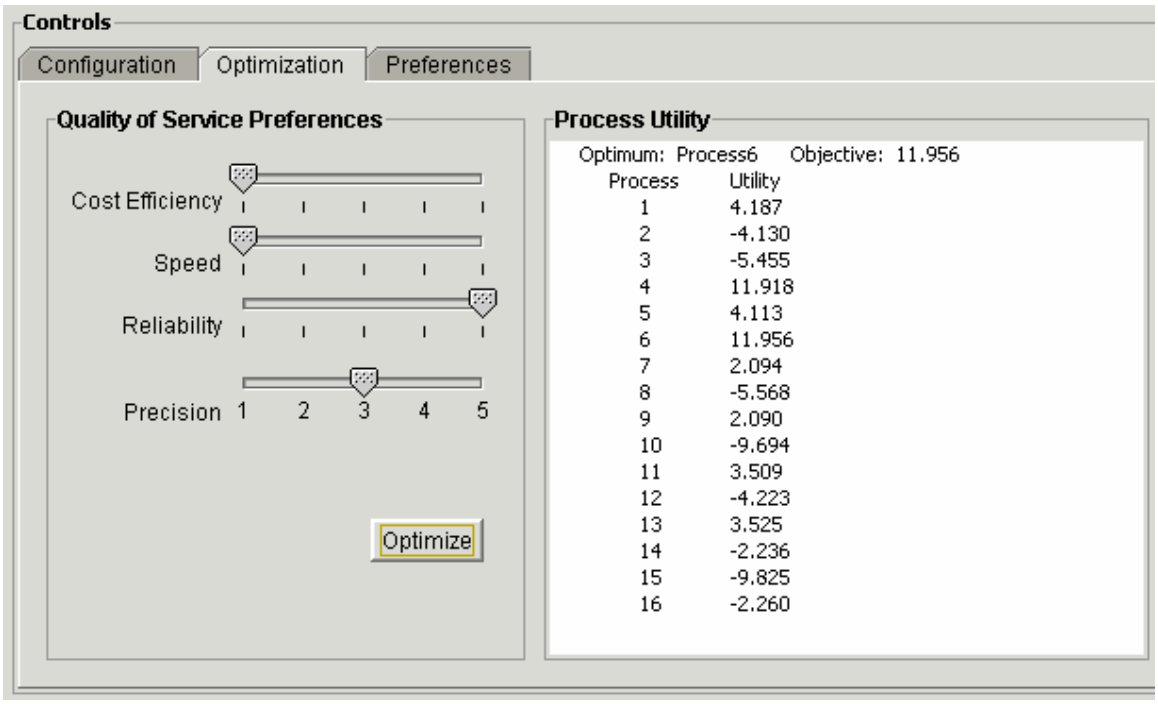

Figure 4.31 Optimizing for reliability

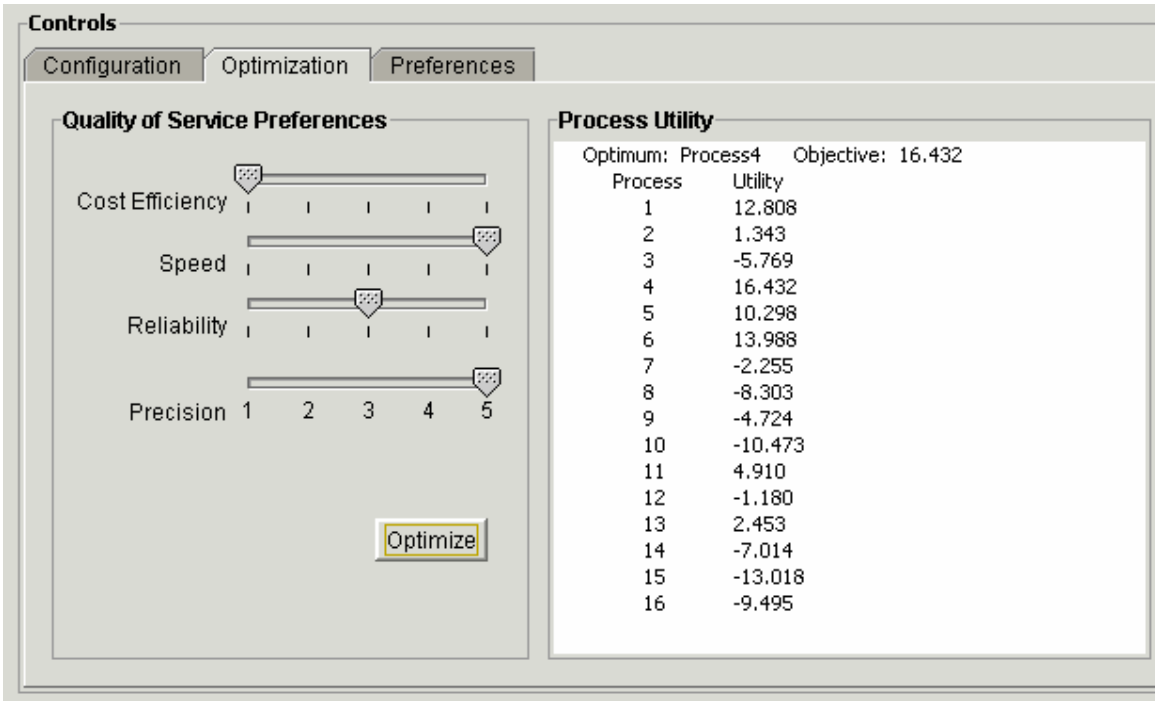

Figure 4.32 Optimizing for multiple objectives

4.5.4 Process Instantiation

Finally, after the optimal process has been configured the user can use the "Controls" panel to save the process or instantiate it into the standard FIPER Design Gateway (see Figure 4.33).

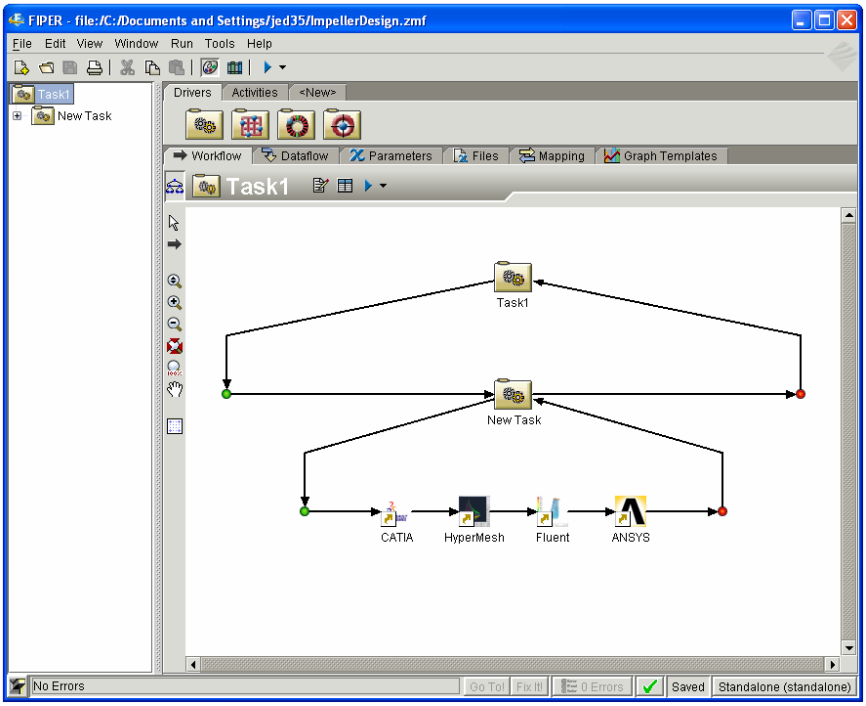

Figure 4.33 Process instantiation into the FIPER Design Gateway

Now a user can utilize the standard FIPER application to make any additional changes to the process as desired and then execute the process in the FIPER Runtime Gateway. Alternately, the user could publish the new process as a service back into the FIPER library. In this way the implementation supports flexible creation of composite services and process configuration at multiple levels of hierarchy.

To demonstrate the flexibility of this approach the newly instantiate abstract process was then configured with product data for a particular impeller design and then executed using the FIPER Runtime Gateway (see Figure 4.34).

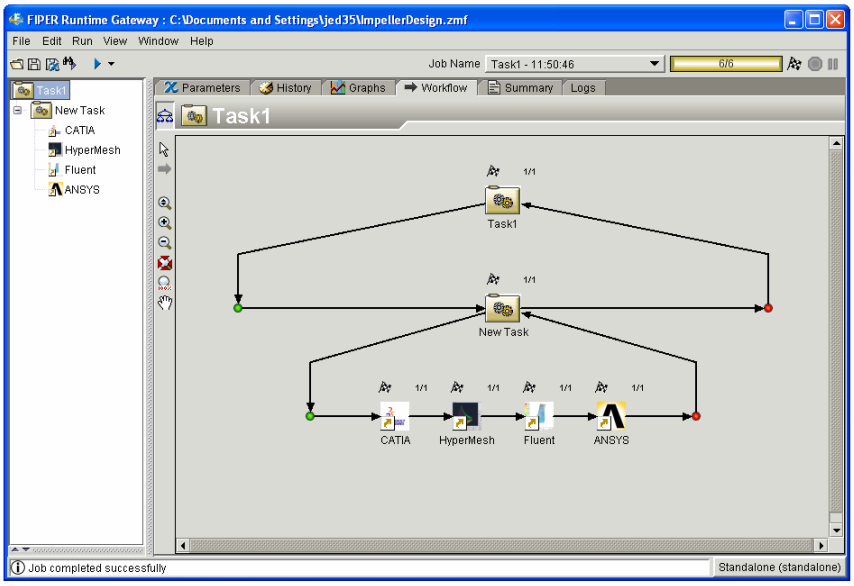

Figure 4.34 Process execution in the FIPER Runtime Gateway

The execution of each service is nearly identical to the execution described by Karpowitz in his impeller design example [88], except that in this case the FIPER application handles all execution details. As described by Karpowitz the CATIA service takes user inputs for master parameters and updates a parametric impeller model of the impeller in CATIA V5. Once the model is updated, IGES (a neutral data format) files for a structural wedge and air solid wedge are exported (see Figure 4.35).

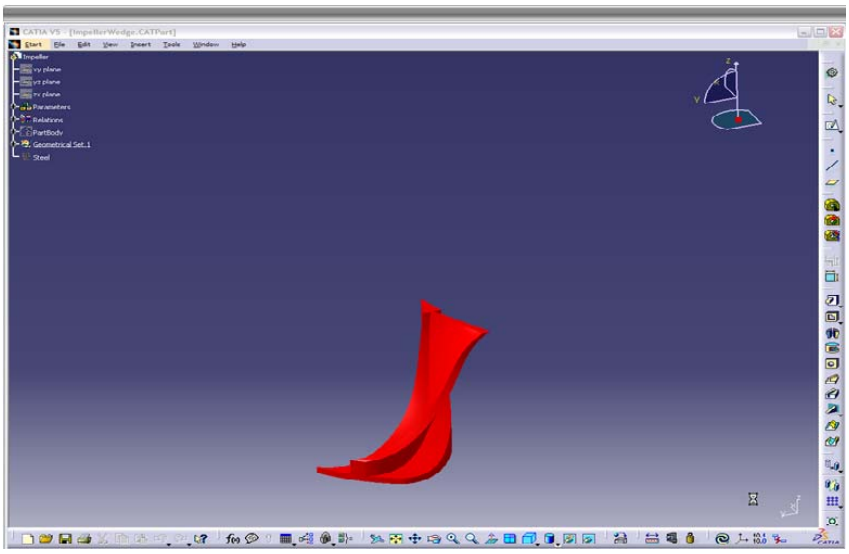

Figure 4.35 The CATIA service updates the impeller blade and exports IGES files

Following execution of the CATIA service, the HyperMesh service uses the air solid IGES file to build surface and volume meshes for finite element calculations as shown in Figure 4.36.

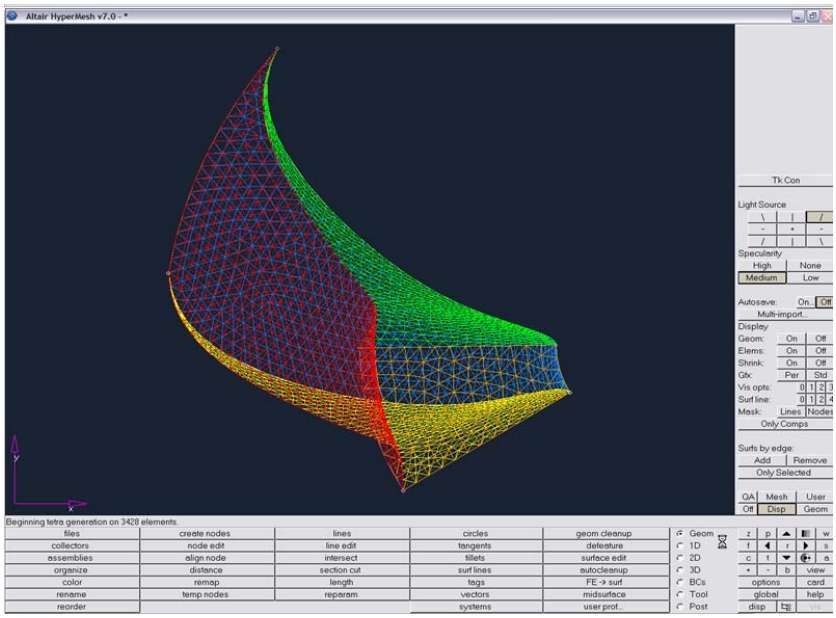

Figure 4.36 The HyperMesh service creates meshes for finite element calculations

Next, the FLUENT service 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 exported in a text file. Additionally, a pressure distribution image is also exported (see Figure 4.37).

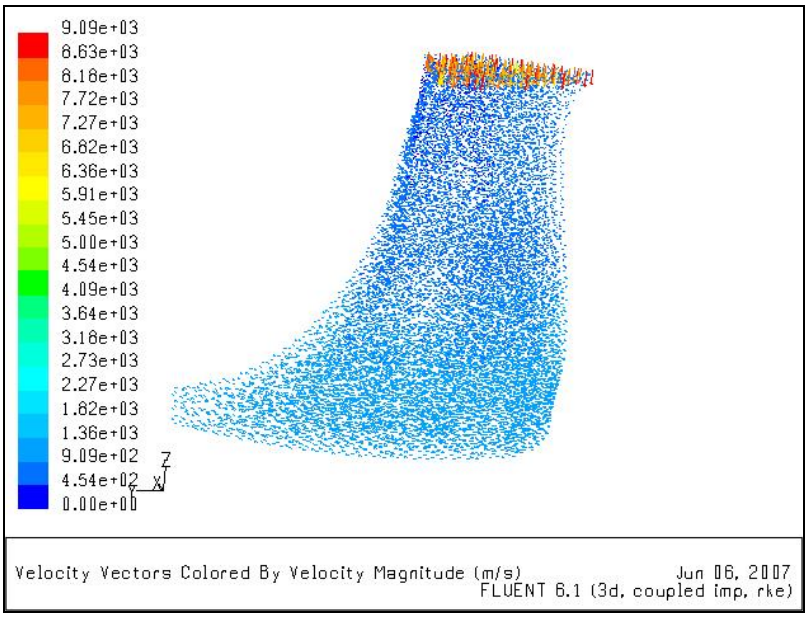

Figure 4.37 The FLUENT service determines pressure on the impeller blade surfaces

Finally, the ANSYS service takes the data from the FLUENT service and the structural IGES file and computes stress values for the structural wedge. This information is exported as a text file and also as an image as shown in Figure 4.38.

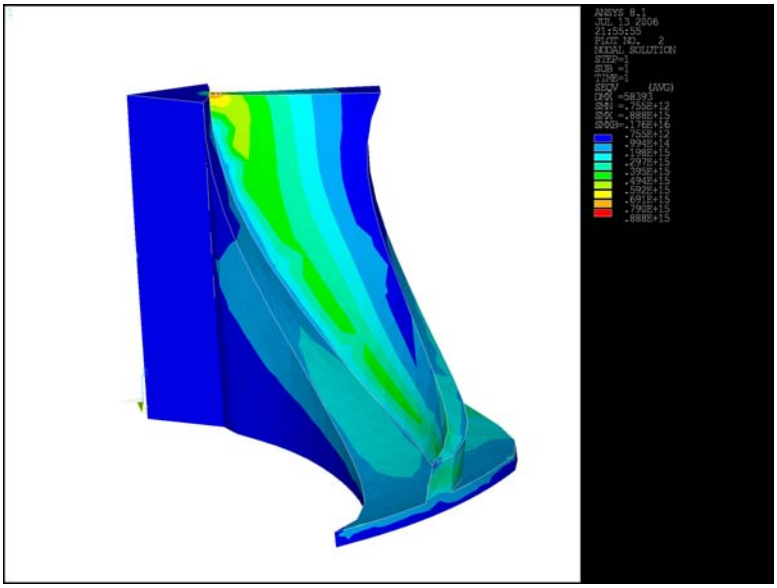

Figure 4.38 The ANSYS service creates stress plots

Besides executing the process each of the services were also published back into the FIPER library (see Figure 4.39). By republishing, these new service implementations become available for dynamic configuration and immediate execution.

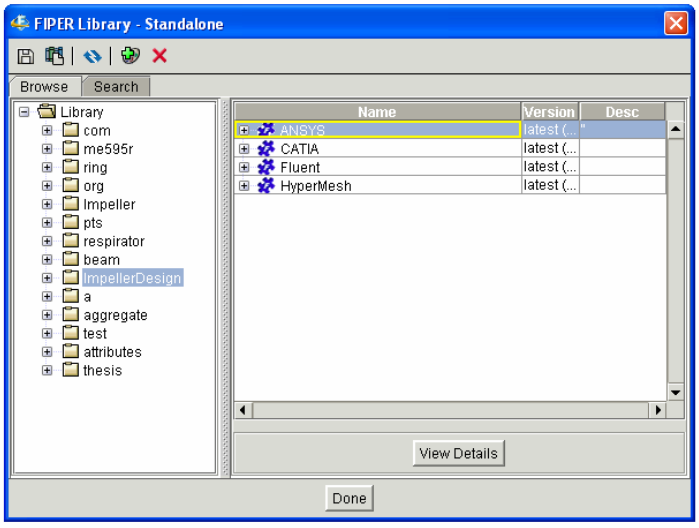

Figure 4.39 Republishing services into the FIPER library

4.6 Process Modeling Examples

Several additional examples demonstrate the ability of the process modeling algorithm to efficiently reconfigure the process in a variety of situations.

4.6.1 Changing Fidelity

It is common in product design to analyze the product at increasingly higher levels of fidelity as more and more product information is created. In the impeller design example, the process modeling algorithm identified all alternative processes that could provide the specified functionality, namely "Structural_Static_3D_MAX_STRESS." Using the engineering analysis ontology, this query explicitly requires a process that can provide a three dimensional fidelity analysis. Specifying a slightly different query of "Structural_Static_2D_MAX_STRESS" produces an entirely different process at a lower fidelity as shown below. This demonstrates the ability of the system to reconfigure itself for a new requirement such as fidelity.

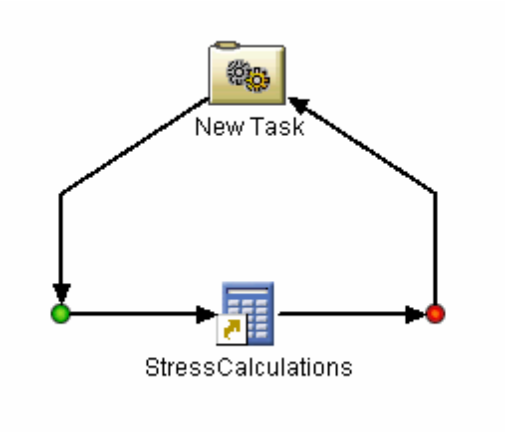

Figure 4.40 Lower fidelity analysis

4.6.2 Parallel Paths, Multiple Queries, Complex Dataflow

The impeller design process example showed a sequential workflow, single query, and only a few dataflow mappings. However, the process modeling algorithm can handle parallel paths, multiple queries, and more dataflow mappings. This complexity can be demonstrated using the PTS (Product Transformation Schematic) developed by Roach et al [22]. As described previously, the PTS is a graph-based representation for designing a product. Using FIPER, the multiple mappings defined by the PTS were implemented as models and published into the FIPER library as shown below.

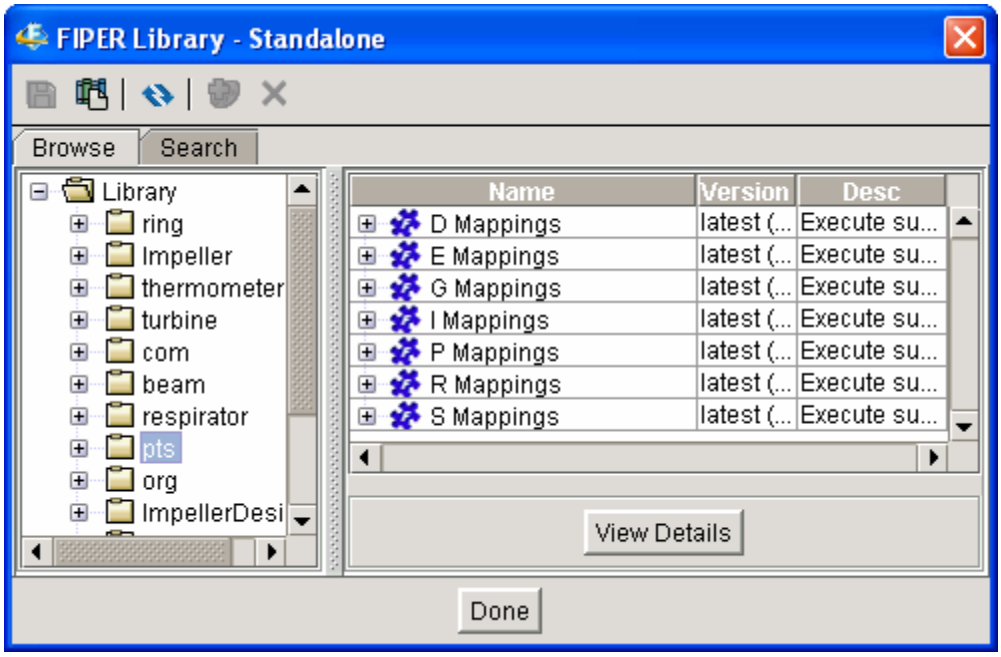

Figure 4.41 PTS models

Multiple queries were then submitted to the FIPER Process Manager. Figure 4.42 through Figure 4.46 show the results from the dynamic configuration. These results demonstrate parallel paths, multiple queries, and complex dataflow.

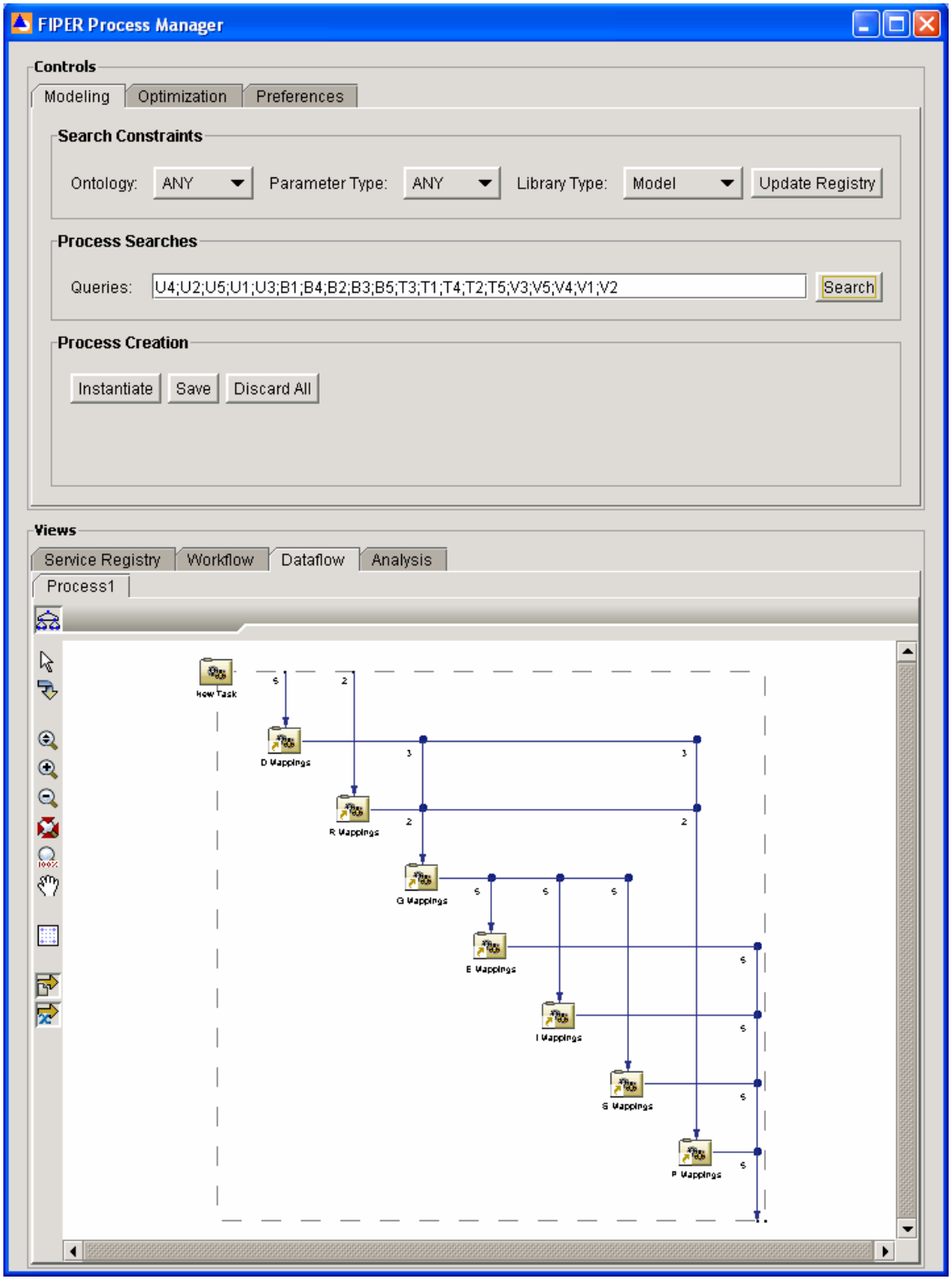

Figure 4.42 PTS dynamic configuration

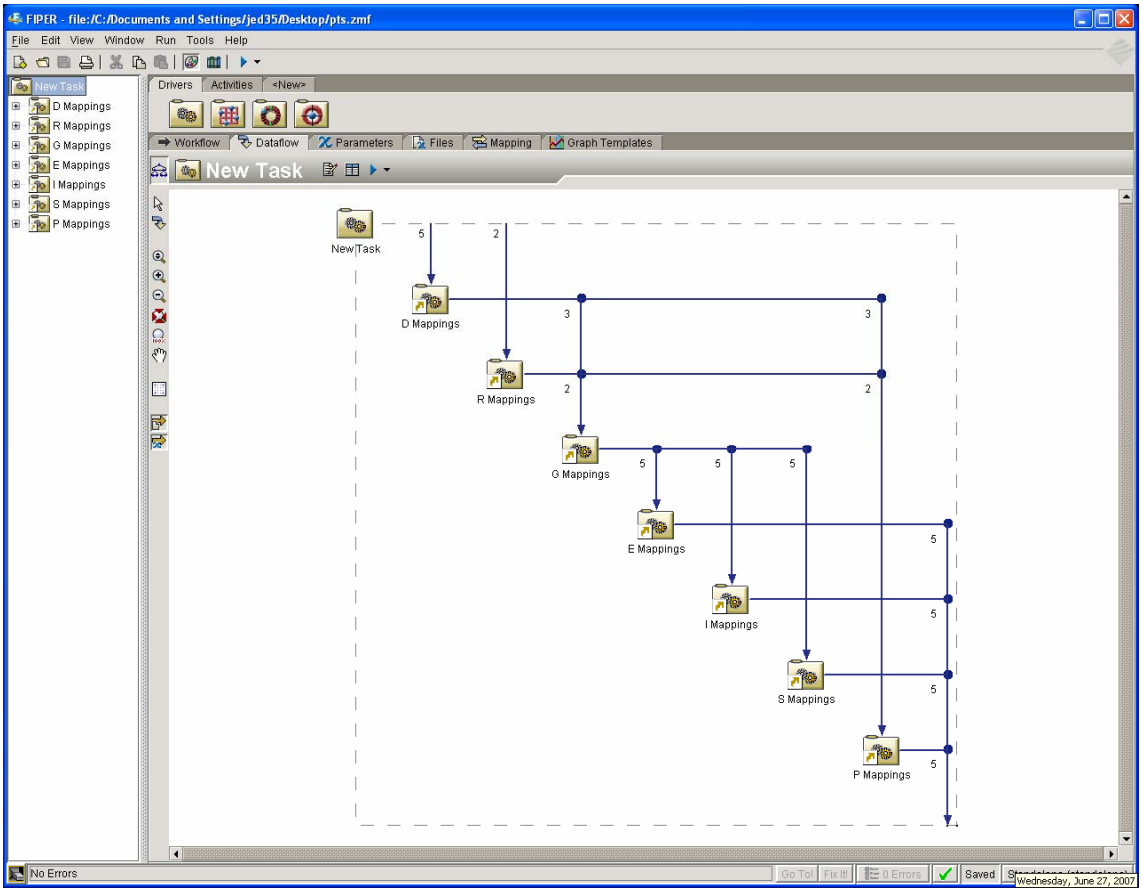

Figure 4.43 PTS complex dataflow

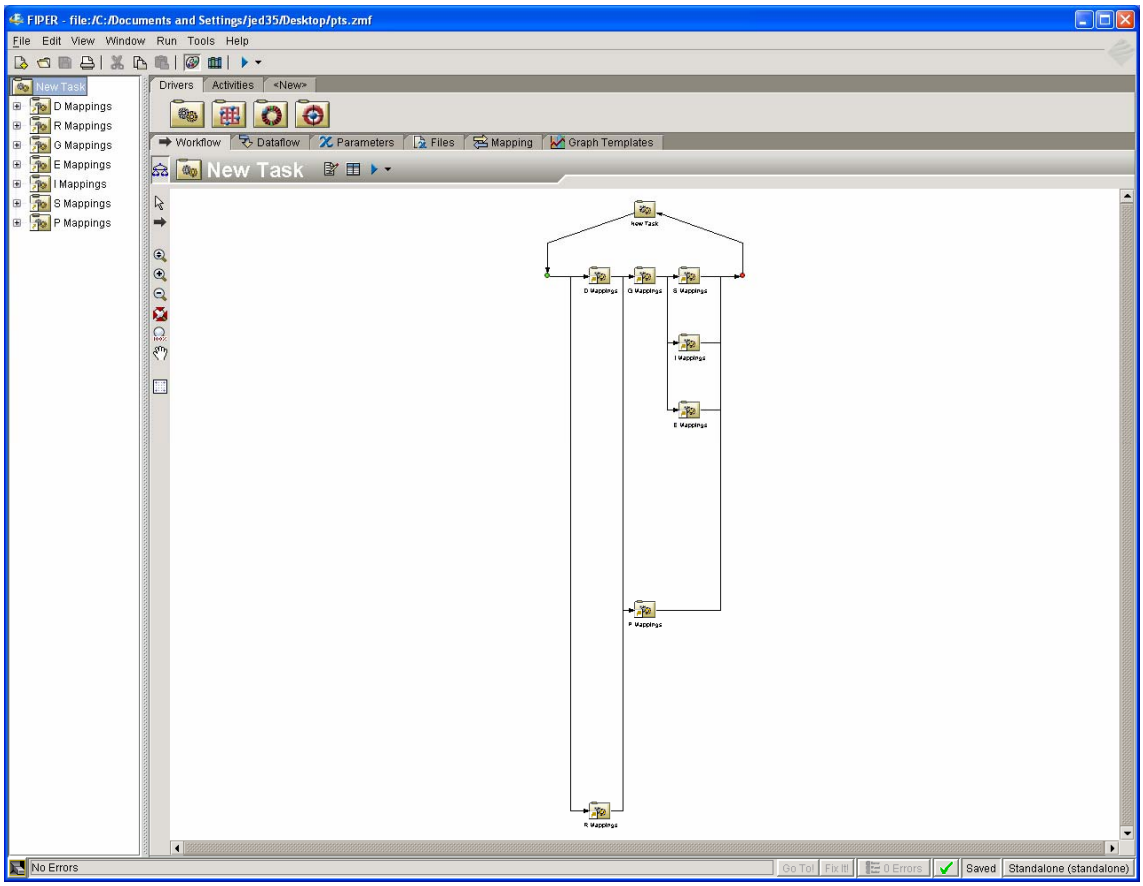

Figure 4.44 PTS parallel paths

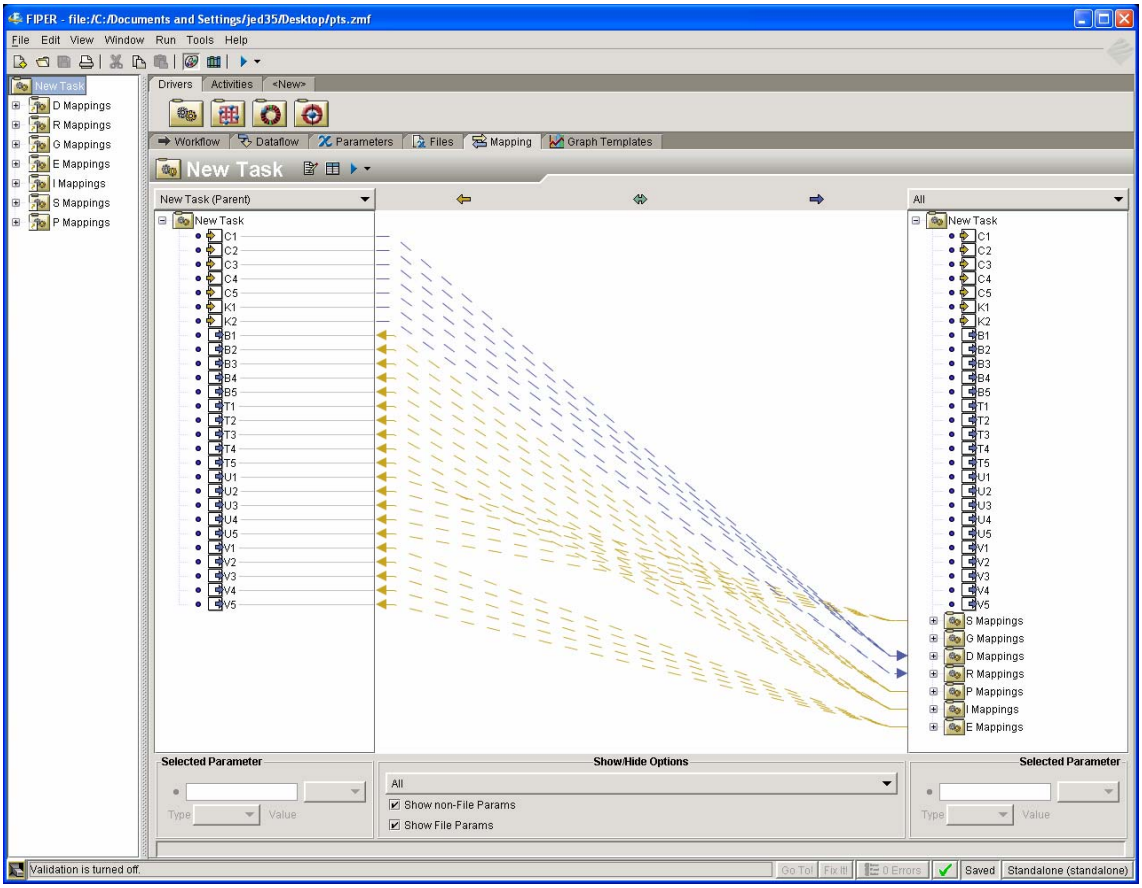

Figure 4.45 PTS multiple queries

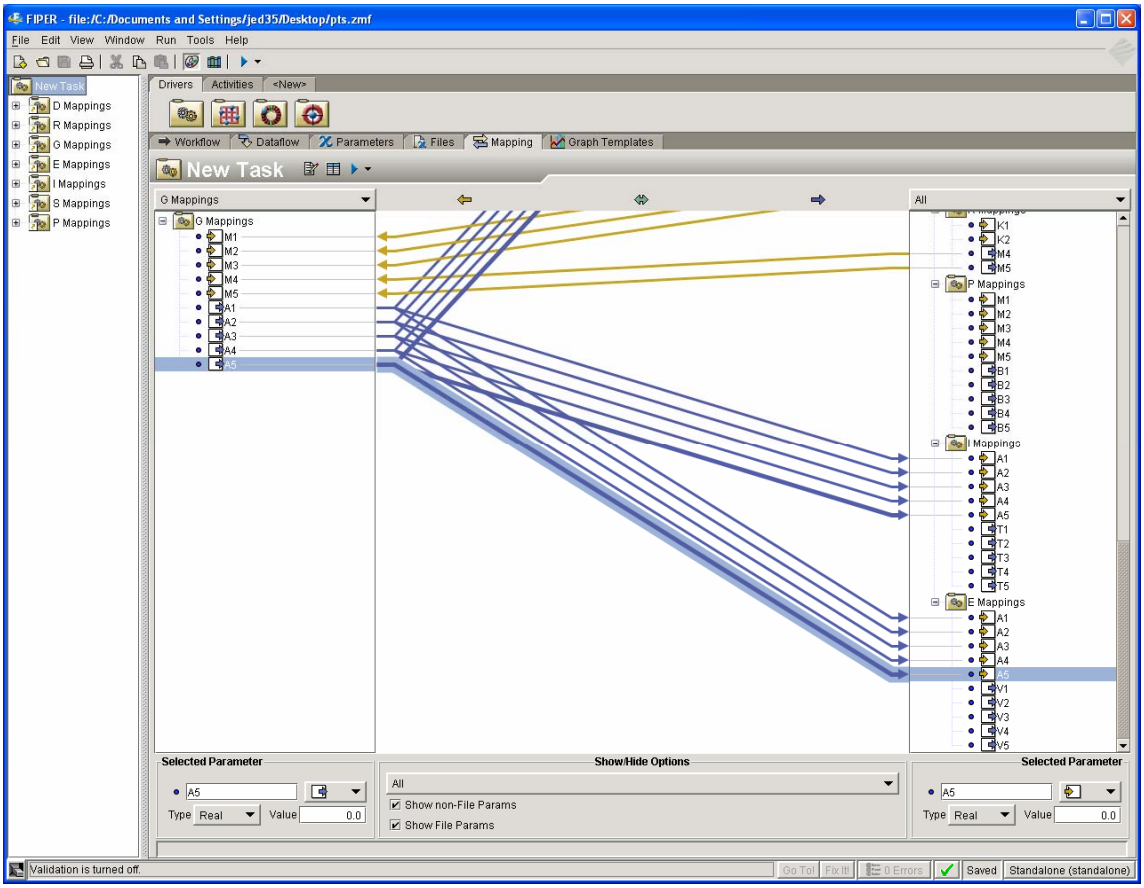

Figure 4.46 PTS multiple dataflow mappings

4.6.3 Hierarchy

Although complex, the PTS example is somewhat abstract. A more concrete implementation of the PTS was developed for a consumer product. The design of a thermometer (see Figure 4.47) was decomposed using the PTS methodology. The thermometer design can be customized for each customer based on the customer's preferences for the shape of the casing, the sophistication of the electronics, and the responsiveness of the heat sensor. The PTS representation describes how design artifacts

such as CAD models, drawings, process plans, and other technical documentation are parameterized as templates that can then be updated based on specific customer inputs.

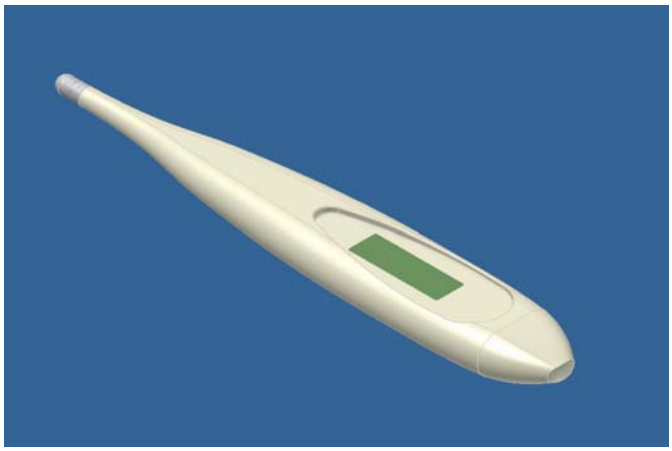

Figure 4.47 Thermometer

The PTS mappings were implemented as FIPER models as shown in Figure 4.48 and published into the library.

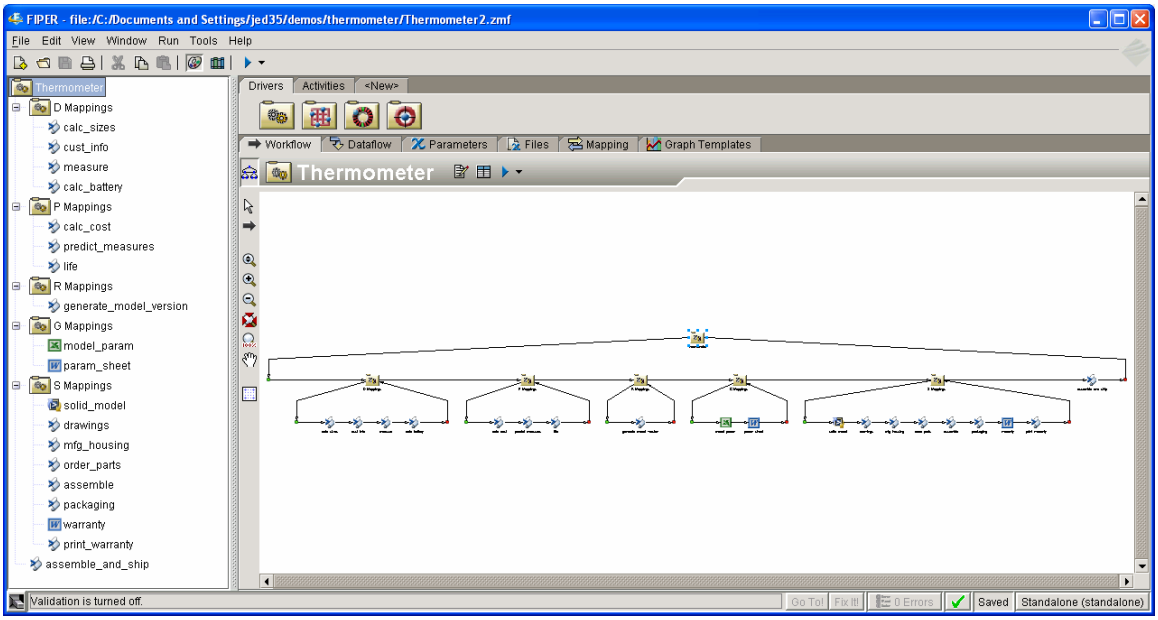

Figure 4.48 Thermometer design process

A query for the completely designed thermometer was then submitted to the FIPER Process Manager. Figure 4.49 shows the resultant process. This figure is very similar to Figure 4.43 from the PTS example because the PTS mappings (D, P, G, R, and S) were used for the thermometer process. Within each of these PTS mappings, the thermometer has product-specific sub-mappings. Figure 4.50 shows an expanded view of the S mapping for the thermometer process. This example demonstrates how the system can be used to dynamically configure a process at different levels of hierarchy.

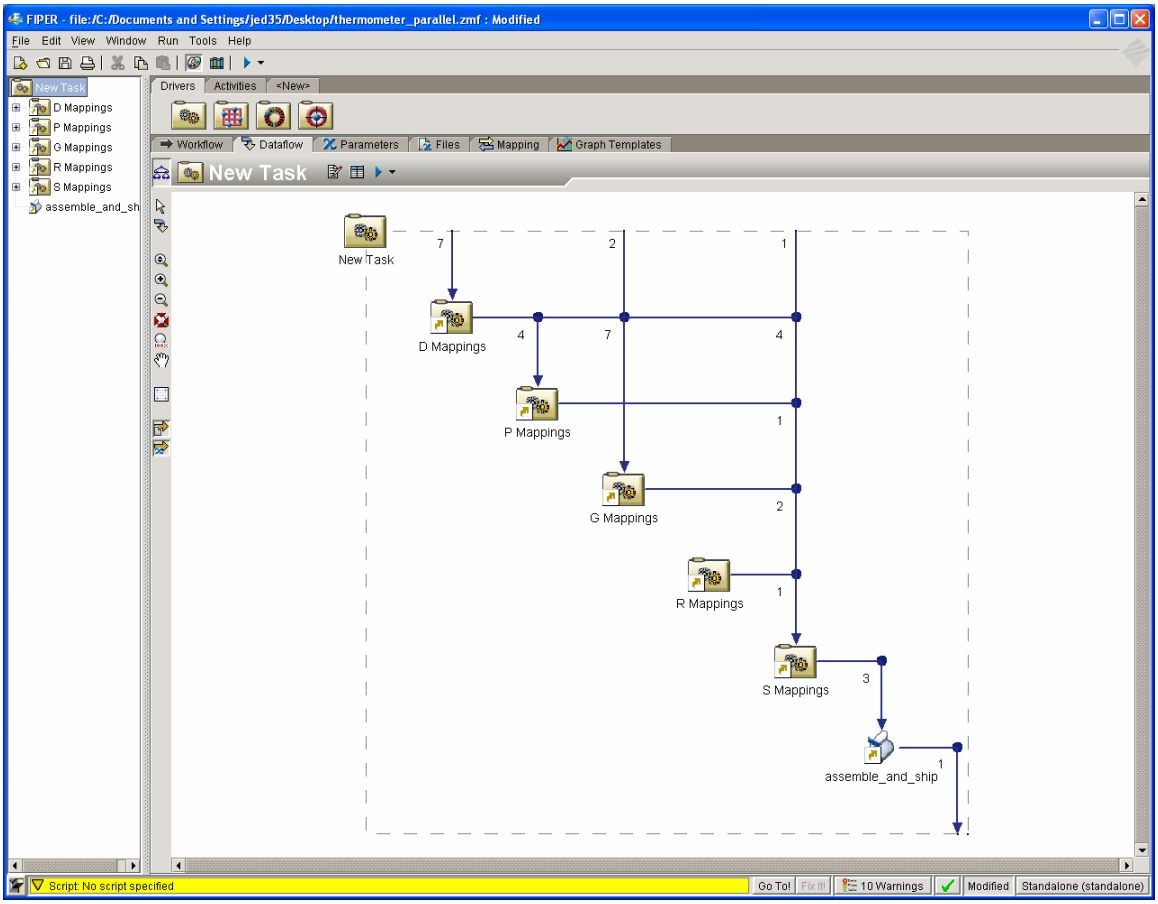

Figure 4.49 Dynamically configured thermometer process

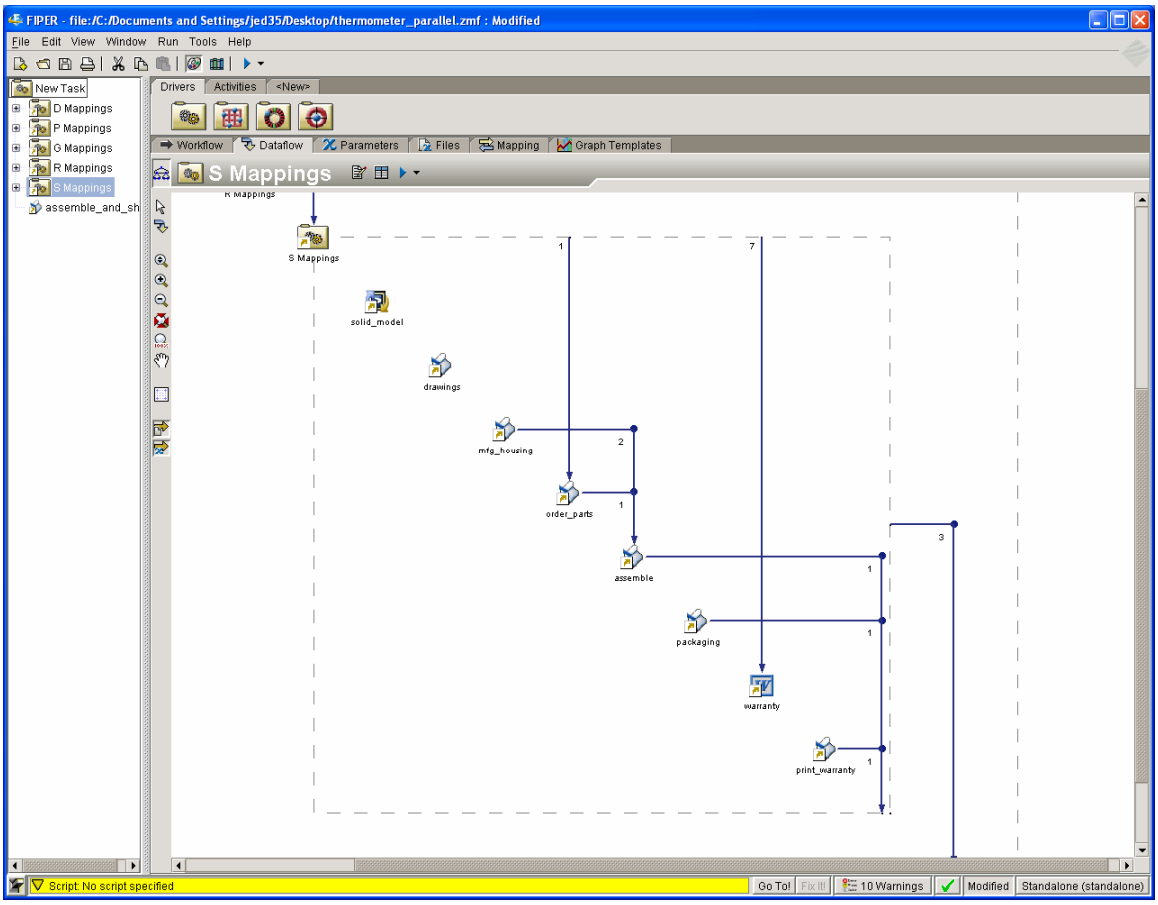

Figure 4.50 Thermometer sub-process

4.6.4 Pruning

A final example demonstrates the ability of the system to efficiently create a sub-process from a larger process. The PTS methodology was again used to decompose a theoretical product. The respirator is a consumer product that helps asthmatic patients on cold days. It consists of an apparatus to fit against the mouth and a heating element which warms cold air as the patient forces air through it during normal breathing. The PTS representation of the respirator design process was implemented as one large FIPER model as shown in Figure 4.51. As in the previous example, this process produces

multiple design artifacts including CAD models and various technical documents. Additionally, it performs various analytical predictions such as cost and life.

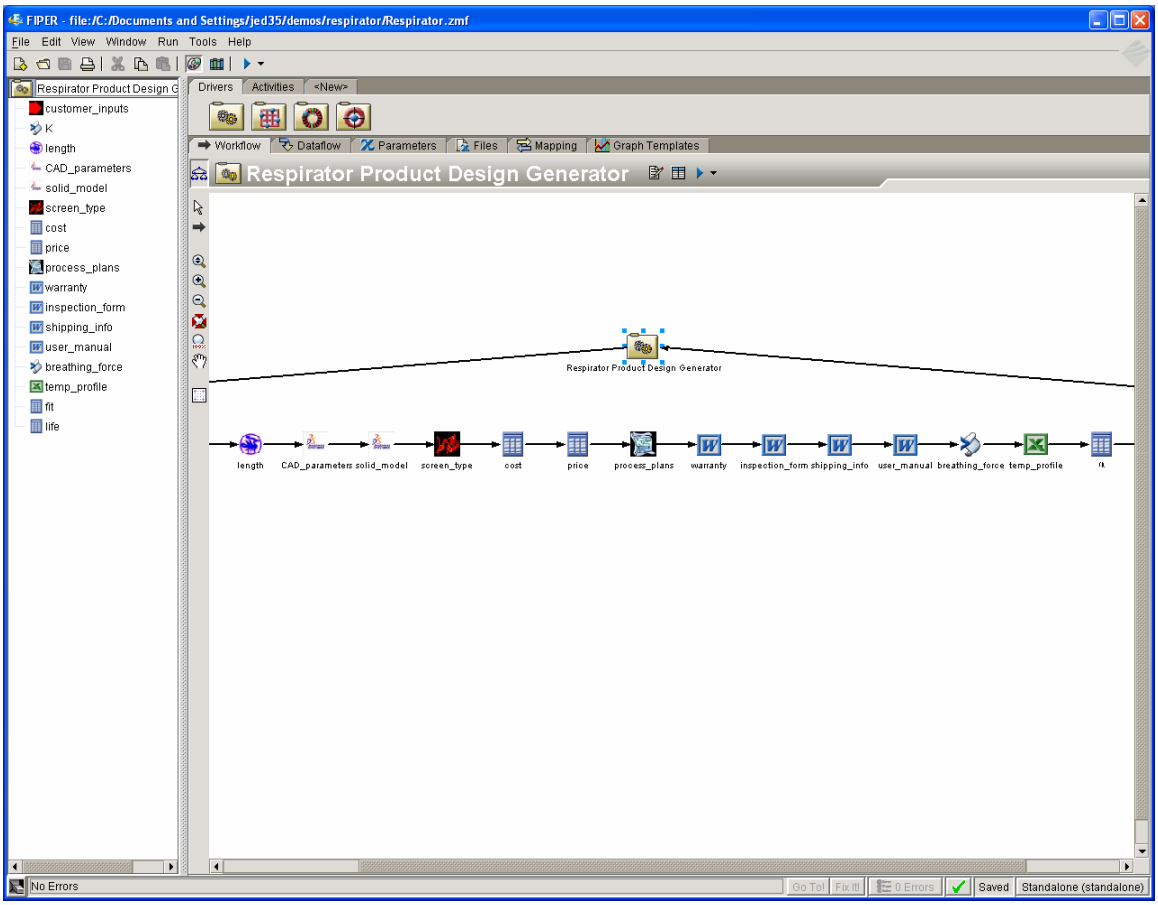

Figure 4.51 Respirator design process

Individual sub-models from this larger model were published into the FIPER library. Then, instead of querying the system for the entire design process, two analyses—cost and life—were submitted as queries with the results shown below. These results demonstrate how a sub-process can be efficiently constructed from a larger process definition based on the desired objectives. This is similar to pruning the larger process to arrive at the sub-process except in this case a bottoms-up approach is used.

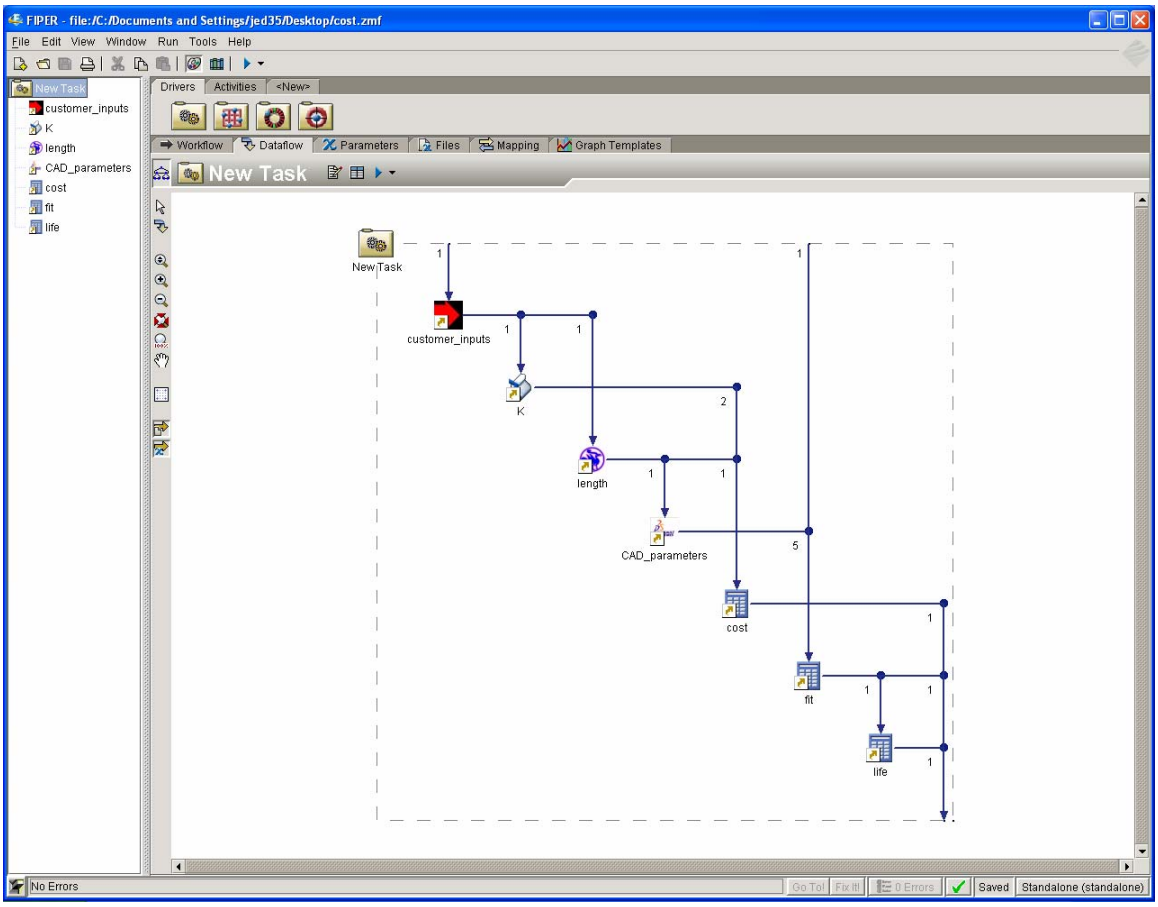

Figure 4.52 Respirator sub-process dataflow

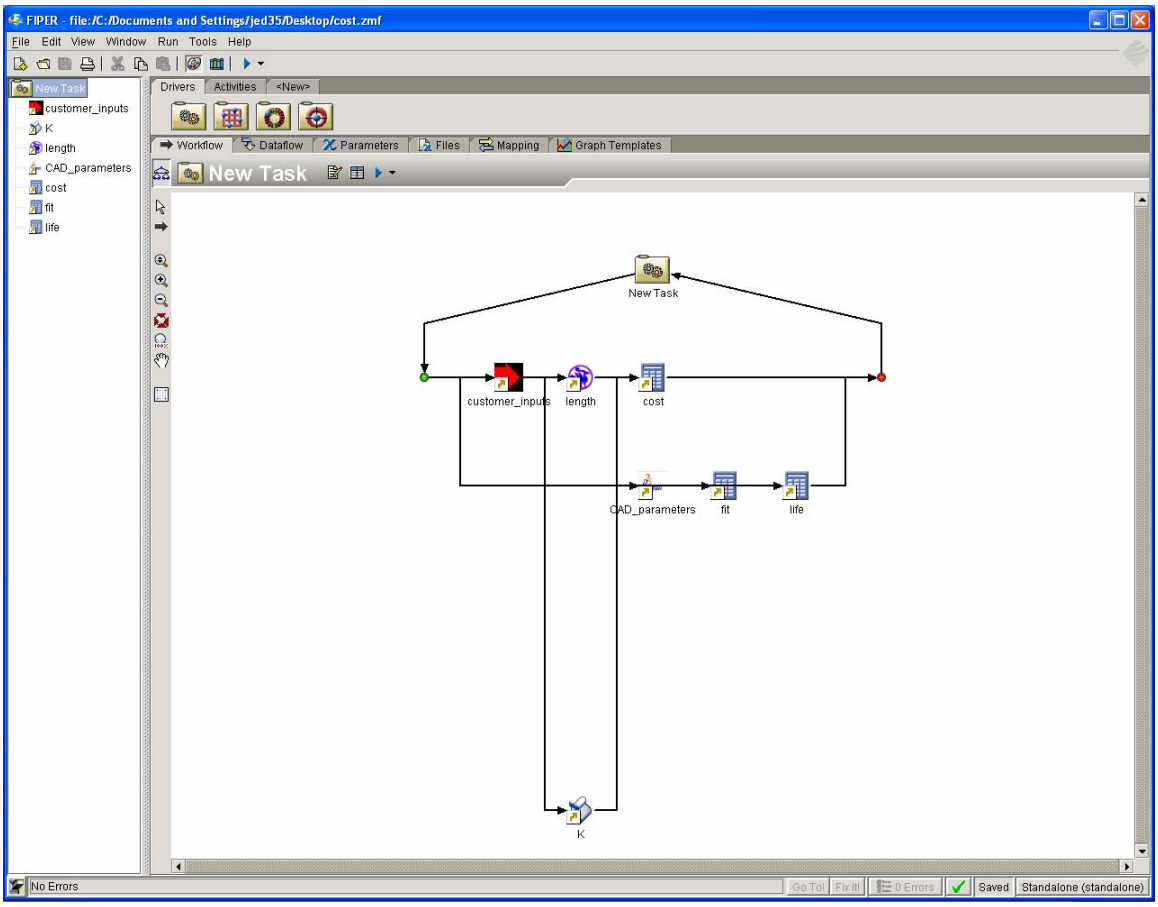

Figure 4.53 Respirator sub-process workflow

4.7 Benefits

The implementation outlined above provides several key benefits in automating product development processes. First, the decomposition of processes into reusable services increases automation efficiency because a standard template is available for a new implementation instead of rewriting code.

Additionally, the process modeling algorithm provides a dynamic approach to configuring processes. These processes are streamlined to provide the desired functionality using concurrent paths and deterministic routings.

Two other benefits can be seen in the differences between the impeller design example presented in this chapter and the example shown by Karpowitz. First, in this work higher level functionality of the services has been captured through using abstract parameters and an engineering analysis ontology. This provides an effective means to dynamically identify alternate services and processes. Second, in this work a QoS ontology implementation enables process comparisons and optimization.

Because the implementation allows a user to rapidly reconfigure automated processes as new services are introduced or as requirements or technologies change, the implementation provides needed flexibility in the dynamic product development environment.

5 Related Work

Because this work is broad in its attempt to provide an overarching approach to more effective product development process automation, there is no direct mapping between this and any previous work. However, much related work has been done on key components of this work including dynamic service discovery, selection, and to a much lesser degree, process optimization. The most relevant of this related work includes research by Karpowitz; Zeng; Maximilien, and Singh. Karpowitz's work is the most closely related and applies directly to product development, so it will be discussed first. The work by Zeng is the next closely related and will be discussed second followed by that of Maximilien and Singh.

5.1 Dynamic Workflow

Karpowitz presented a dynamic workflow framework for implementing product development process automation involving web services and software agents [8]. Within this framework, engineering processes are decomposed into specific tasks which are then individually wrapped in a Web service. A service description is then provided to a registry management agent that publishes the service to a registry. A knowledge agent then uses the registry to call the standard reporting methods from the new service to update the registry with additional descriptive information (e.g. inputs and outputs).

Once the service registry is populated, a specific desired output can be provided to the system through a web browser. A configuration agent then determines the sequence (or sequences) of service execution that can produce the desired output. This agent can also grade each sequence based on some predetermined criteria. Finally, an execution agent can choose the process sequence with the best grade and execute the sequence.

The system demonstrated that engineering processes can be decomposed into web services. Additionally, it showed that simple services can be combined dynamically to produce a specific output.

This work has attempted to improve on Karpowitz's framework in the following ways. First, the approach implements a new process modeling algorithm to include bifurcation and multiple queries. A key aspect of the algorithm is its focus on dataflow. Whereas the algorithm used by Karpowitz is primarily focused on building process sequences, the dataflow focus of the new algorithm enables concurrent process flows and multiple dataflow connections for more complex processes.

Second, this work defines functional descriptions for the services based on a product development ontology to enable dynamic service discovery. These higher level descriptions provide a more manageable ontological approach than simple name matching on low level parameters.

Third, nonfunctional descriptions for the services based on a Quality of Service (QoS) ontology are defined as well as a method to evaluate overall process utility. This enables process comparisons and optimization to use resources more effectively within an enterprise.

Fourth, this approach was implemented on top of a commercial product development process integration software (FIPER) in order to facilitate human interaction with the system and make use of a more sophisticated process modeling language to support more complex processes.

5.2 Process Optimization

Zeng et al [27] focus their research on selecting Web services for composition into larger services. As in this work, their's selects an optimal process based on a utility function. This utility function takes into account both QoS and user preferences. Additionally, they provide several aggregation functions as in this work in order to evaluate QoS of entire processes. They provide a rigorous mathematical approach to the optimization problem and present two methods for optimization—local and global. In the local method service selection decisions are based on each individual service QoS. In the global method services are only selected after considering aggregated QoS metrics across the entire process.

Their work is complementary to this work because their focus is runtime selection and the focus of this work is design time selection. Both works present different QoS aggregation functions that could be used together to support further QoS extensions.

The major difference between their work and this work is their optimization problem at runtime versus this work's optimization problem at design time. In the former case, they provide an extensive definition of a graph representation of an execution path. Nodes on the graph represent tasks that can be executed by a set of candidate services based on functional requirements. They define an execution plan as the execution of the

execution path after selecting a single candidate service at each node. In this work each task can also be executed by a set of candidate services. However, candidate services may also include a composite of several services that are dynamically composed during the process modeling stage. In other words, in this work optimization is done across an execution path with expandable nodes. On the other hand their work is more sophisticated in including conditional paths to support runtime decision making.

Additionally, this work focuses on product development specifics such as ontological frameworks for functional descriptions and QoS metrics for engineering analysis services.

5.3 Service Selection Using Quality of Service

Maximilien and Singh have provided a significant amount of research in the use of multiagent systems for dynamically selecting web services. They have developed a full ontological representation for QoS [26] that can be extended into specific domains such as product development. They have also developed trust and reputation models for using Web services and a method for agents to dynamically monitor and adjust quality ratings [31] [32]. Additionally, they have integrated these theories into full multiagent system implementations [15] [33].

Their work has impacted this work in a variety ways. First, this work uses their QoS ontology for describing nonfunctional service attributes [26]. Several quality metrics were subclassed from their middle ontology in order to provide an example for engineering analysis.

Second, concepts from their trust model [33] were used to develop the process utility equation used in this work. However, the formulation used here was much less sophisticated, and did not incorporate trust and reputation factors. Future work could include these factors as well as their multiagent approaches in this regard. Such future work would provide a mechanism for dynamically monitoring and certifying QoS attributes whereas the current implementation relies on central management by a human user.

Third, their work makes use of various multiagent system implementations. This work does not make use of a multiagent system implementation. The implementation in the current work relies on human users and a single service registry. Some experimentation was done using the Java Agent Development Environment (JADE) to perform process configuration queries across a network. However, this experimentation turned out to be somewhat peripheral to this work.

In the future such multi-agent implementations may become important as the framework is extended to operate in more open environments and additional concepts of self-management and autonomic systems are supported. Additionally, agent collaboration may enable autonomic product development design once the current work on autonomic product development processes is complemented by autonomic product data management.

Finally, it is important to point out that a significant difference between this work and their work is that the focus of this work is on creating larger processes using Web services and then selecting optimal processes at design time whereas their work focuses on selecting optimal services during runtime. This design time perspective for product

development is an extremely important preliminary to any future runtime research because it complements the current role of human designers who will want, at least initially, to have a static runtime process in order to evaluate the effectiveness of the proposed automation framework.

6 Conclusion

The framework presented in this work provides a flexible, standardized approach to product development process automation. Key elements of the framework include a formal approach to decompose product development processes into reusable services, a method to describe functional and quality attributes of services, a process modeling algorithm to configure processes composed of services, a method to evaluate process utility based on quality metrics and user preferences, and an implementation that allows a user to instantiate the optimal process.

Significant contributions to improve product development process automation effectiveness can be described as follows. First, the decomposition of processes into reusable services increases automation efficiency because a standard template is available for a new automation implementation. Thus software maintenance is reduced for new automation projects because less new code is required.

Second, the process modeling algorithm provides a dynamic approach to configuring processes. The algorithm streamlines these processes to provide the desired functionality using concurrent paths and deterministic routings.

Third, in this work higher level functionality of the services has been captured through using abstract parameters and an engineering analysis ontology. This provides an effective means to dynamically identify alternate processes. These alternates provide

flexible process definitions for an enterprise and reusable templates available for implementation.

Fourth, in this work a QoS ontology implementation enables process comparisons and optimization. This QoS ontology also facilitates better process benchmarking and ensures that best practices are used within an organization.

Fifth, this approach provides a means to instantiate and modify processes, thus complementing the role of a human user and providing a model driven design to the process automation activity.

Finally, a process automation activity has been demonstrated for a realistic engineering analysis, thereby demonstrating the flexibility and merits of the approach within an industry context.

The current framework should be tested within a commercial enterprise before the scalability and effectiveness of the approach can be proved. Future work may also include increased self-management using agent-based frameworks. Such self-management tasks would include dynamic monitoring and certifying of QoS attributes, mapping parameters across multiple ontologies, and runtime support.

7 References

[1] IBM. “New to SOA and Web Services.” http://www-128.ibm.com/developerworks/webservices/newto/, (accessed 15 May 2006).

[2] IBM. “Transforming Legacy Applications Into An SOA Framework.” ftp://ftp.software.ibm.com/software/solutions/pdfs/Transforming_legacy_applications_into_an_SOA_framework.pdf, (accessed 15 May 2006).

[3] Web Services Interoperability Organization. “WS-I Basic Profile Version 1.1.” http://www.ws-i.org/Profiles/BasicProfile-1.1-2004-08-24.html, (accessed 15 May 2006).

[4] IBM. “The Vision of Autonomic Computing.” http://researchweb.watson.ibm.com/autonomic/research/papers/AC_Vision_Computer_Jan_2003.pdf, (accessed 16 May 2006).

[5] IBM. “The Eight Elements.” http://researchweb.watson.ibm.com/autonomic/overview/elements.html, (accessed 16 May 2006).

[6] Lander, S.E. “Issues in Multiagent Design Systems.” *IEEE Expert* 12, no. 2 (1997): 18-26.

[7] Wooldridge, M. *An Introduction to MultiAgent Systems*. New York: John Wiley & Sons, 2002.

[8] Karpowitz, D.J. “A Dynamic Workflow Framework for Mass Customization Using Web Service and Autonomous Agent Technologies.” M.S. thesis, Brigham Young University, 2006.

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

[10] Campbell, M.I., J. Cagan, and K. Kotovsky. “A-Design: An Agent-based Approach to Conceptual Design in a Dynamic Environment.” *Research in Engineering Design* 11 (1999): 172–192.

[11] Birmingham, W.P., and T.P. Darr. “Automated Design for Concurrent Engineering.” *IEEE Expert* 9, no. 5, (1994): 35-42.

[12] Sun, J., Y.F. Zhang, 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.

[13] Shen, W., D.H. Norrie, and J.P.A. Barthes. *Multi-Agent Systems for Concurrent Intelligent Design and Manufacturing*. London: Taylor and Francis, 2001.

[14] Buhler, P.A., J.M. Vidal, and H. Verhagen. “Adaptive Workflow = Web Services + Agents.” *Proceedings of the International Conference on Web Services, Las Vegas, NV, 2003. 131-17. CSREA Press. 2003)*

[15] 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*.

[16] 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*.

[17] World Wide Web Consortium. “Semantic Web.” http://www.w3.org/2001/sw/, (accessed 17 May 2007).

[18] Palmer, S.B. “The Semantic Web: An Introduction.” http://infomesh.net/2001/swintro/, (accessed 17 May 2007).

[19] World Wide Web Consortium. “OWL Web Ontology Language Guide.” http://www.w3.org/TR/owl-guide/, (accessed 17 May 2007).

[20] DAML. “DAML Services.” http://www.daml.org/services/owl-s/, (accessed 17 May 2007).

[21] World Wide Web Consortium. “OWL-S: Semantic Markup for Web Services.” http://www.w3.org/Submission/OWL-S/, (accessed 17 May 2007).

[22] Roach, G.M., J.J. Cox, and C.D. Sorenson. “The Product Design Generator: A System for Producing Design Variants.” *International Journal of Mass Customization* 1, no. 1, (2005): 83-106.

[23] Young, J.M. “Nesting Automated Design Modules in an Interconnected Framework.” M.S. thesis, Brigham Young University, 2005.

[24] Bailey, M.W., and W.H. VerDuin. “FIPER: An Intelligent System for the Optimal Design of Highly Engineered Products.” http://www.engineous.com/resources.htm, (accessed 17 May 2007).

[25] Kao, K.J., C.E. Seeley, S. Yin, and R.M. Kolonay. “Business-to-Business Virtual Collaboration of Aircraft Engine Combustor Design.” In *Proc. of the Design Engineering Technical Conferences, Chicago, Illinois, 2003*.

[26] Maximilien, E.M., and M.P. Singh. “A Framework and Ontology for Dynamic Web Services Selection.” *IEEE Internet Computing* 8, no. 5 (September 2004): 84-93.

[27] Zeng, L., B. Benatallah, A.H.H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. “QoS-Aware Middleware Web Services Composition.” *IEEE Transactions on Software Engineering* 30, no. 5, (May 2004): 311-327.

[28] World Wide Web Consortium. “QoS for Web Services: Requirements and Possible Approaches.” http://www.w3c.or.kr/kr-office/TR/2003/ws-qos/, (accessed 16 May 2007).

[29] Ran, S. “A Model for Web Services Discovery with QoS.” *SIGecom Exchanges* 4, no. 1, (2004): 1-10.

[30] Chatterjee, B.S.S., M.D.J.J. Sydir, and T.F. Lawrence. “Taxonomy for QoS Specification.” *Workshop on Object-Oriented Real-Time Dependable systems (WORDS ’97)*, IEEE CS Press, 1997.

[31] Maximilien, E.M., and M.P. Singh. “Toward Autonomic Web Services Trust and Selection.” In *Proc. of 2nd International Conference on Service-Oriented Computing, New York, New York, November 2004*.

[32] Maximilien, E.M., and M.P. Singh. “Reputation and Endorsement for Web Services.” *SIGecom Exchanges* 3, no. 1, (2001): 1-10.

[33] Maximilien, E.M., and M.P. Singh. “Multiagent System for Dynamic Web Services Selection.” *Workshop on Service-Oriented Computing and Agent-Based Engineering, Ultrecht, Netherlands, July 2005*.

[34] Havey, M. *Essential Business Process Modeling*. Sebastopol, CA: O’Reilly Media, Inc., 2005.

[35] Hillion, S. “Dynamic Java.” http://koala.ilog.fr/djava/, (accessed 22 May 2007).

Appendix A. Engineering Analysis Ontology

```
<ontologies>
  <ontology name="Analysis">
    <discipline name="Geometry">
      <analysisType name="Solid">
        <dimensionality name="3D">
          <condition>MASTER_PARAMETERS</condition>
          <condition>PART</condition>
        </dimensionality>
      </analysisType>
      <analysisType name="Surface"></analysisType>
      <analysisType name="Wireframe"></analysisType>
    </discipline>
    <discipline name="Meshing">
      <analysisType name="Grid">
        <dimensionality name="3D">
          <condition>MESH</condition>
        </dimensionality>
      </analysisType>
    </discipline>
    <discipline name="Fluids">
      <analysisType name="Incompressible">
      </analysisType>
      <analysisType name="Compressible">
        <dimensionality name="3D">
          <condition>TOTAL_PRESSURE</condition>
        </dimensionality>
      </analysisType>
    </discipline>
    <discipline name="Structural">
      <analysisType name="Static">
        <dimensionality name="3D">
          <condition>MAX_STRESS</condition>
        </dimensionality>
      </analysisType>
      <analysisType name="Modal"></analysisType>
      <analysisType name="Creep"></analysisType>
```

```
            <analysisType name="Impact"></analysisType>
            <analysisType name="Fracture"></analysisType>
        </discipline>
        <discipline name="Thermal"></discipline>
        <discipline name="Cost"></discipline>
        <discipline name="Forging"></discipline>
        <discipline name="Other"></discipline>
        <QoS_metrics>
            <QoS name="Precision">
                <QoS name="RSquared">
                    <qValue>
                        <min>0.0</min>
                        <max>1.0</max>
                        <type>real</type>
                    </qValue>
                </QoS>
            </QoS>
            <QoS name="Cost">
                <QoS name="VariableCost">
                    <qValue>
                        <units>USD/hours</units>
                        <type>real</type>
                    </qValue>
                </QoS>
            </QoS>
            <QoS name="Performance">
                <QoS name="ExecutionTime">
                    <qValue>
                        <units>seconds</units>
                        <min>0.0</min>
                        <max>null</max>
                        <type>real</type>
                    </qValue>
                </QoS>
            </QoS>
            <QoS name="Reliability">
                    <qValue>
                        <min>0.0</min>
                        <max>1.0</max>
                        <type>real</type>
                    </qValue>
            </QoS>
        </QoS_metrics>
    </ontology>
</ontologies>
```

Appendix B. Service Descriptions

```
<services>
  <service name="org.apdl.Analysis.Geometry.Solid.3D.Update.CATIA">
    <description>
      <ontology>Analysis</ontology>
      <QoS name="VariableCost">
        <qValue>
          <min>11.75</min>
          <max>11.75</max>
          <typical>11.75</typical>
        </qValue>
      </QoS>
      <QoS name="ExecutionTime">
        <qValue>
          <min>5.0</min>
          <max>30.0</max>
          <typical>19.0</typical>
        </qValue>
      </QoS>
      <QoS name="Reliability">
        <qValue>
          <typical>0.80</typical>
        </qValue>
      </QoS>
      <QoS name="RSquared">
        <qValue>
          <min>1.0</min>
          <max>1.0</max>
          <typical>1.0</typical>
        </qValue>
      </QoS>
    </description>
    <conditions>
<precondition>Analysis.Geometry.Solid.3D.MASTER_PARAMETERS</precondition>
      <postcondition>Analysis.Geometry.Solid.3D.PART</postcondition>
    </conditions>
  </service>
```

```
<service name="org.apdl.Analysis.Meshing.Grid.3D.Hypermesh">
  <description>
    <ontology>Analysis</ontology>
    <QoS name="VariableCost">
      <qValue>
        <min>1.75</min>
        <max>1.75</max>
        <typical>1.75</typical>
      </qValue>
    </QoS>
    <QoS name="ExecutionTime">
      <qValue>
        <min>15.0</min>
        <max>45.0</max>
        <typical>30.0</typical>
      </qValue>
    </QoS>
    <QoS name="Reliability">
      <qValue>
        <typical>0.9</typical>
      </qValue>
    </QoS>
    <QoS name="RSquared">
      <qValue>
        <min>1.0</min>
        <max>1.0</max>
        <typical>1.0</typical>
      </qValue>
    </QoS>
  </description>
  <conditions>
    <precondition>Analysis.Geometry.Solid.3D.PART</precondition>
    <postcondition>Analysis.Meshing.Grid.3D.MESH</postcondition>
  </conditions>
</service>
<service name="org.apdl.Analysis.Fluids.Compressible.3D.Fluent">
  <description>
    <ontology>Analysis</ontology>
    <QoS name="VariableCost">
      <qValue>
        <min>1.75</min>
        <max>1.75</max>
        <typical>1.75</typical>
      </qValue>
    </QoS>
    <QoS name="ExecutionTime">
```

```
                        <qValue>
                            <min>20.0</min>
                            <max>1000.0</max>
                            <typical>90.0</typical>
                        </qValue>
                    </QoS>
                    <QoS name="Reliability">
                        <qValue>
                            <typical>0.95</typical>
                        </qValue>
                    </QoS>
                    <QoS name="RSquared">
                        <qValue>
                            <min>0.50</min>
                            <max>0.95</max>
                            <typical>0.87</typical>
                        </qValue>
                    </QoS>
                </description>
                <conditions>
                    <precondition>Analysis.Meshing.Grid.3D.MESH</precondition>
<postcondition>Analysis.Fluids.Compressible.3D.TOTAL_PRESSURE</postcondition>
                </conditions>
            </service>
            <service name="org.apdl.Analysis.Structural.Static.3D.Ansys">
                <description>
                    <ontology>Analysis</ontology>
                    <QoS name="VariableCost">
                        <qValue>
                            <min>3.75</min>
                            <max>3.75</max>
                            <typical>3.75</typical>
                        </qValue>
                    </QoS>
                    <QoS name="ExecutionTime">
                        <qValue>
                            <min>10.0</min>
                            <max>50.0</max>
                            <typical>20.0</typical>
                        </qValue>
                    </QoS>
                    <QoS name="Reliability">
                        <qValue>
                            <typical>0.95</typical>
                        </qValue>
                    </QoS>
```

```
                <QoS name="RSquared">
                  <qValue>
                    <min>0.50</min>
                    <max>0.98</max>
                    <typical>0.91</typical>
                  </qValue>
                </QoS>
              </description>
              <conditions>
                <precondition>Analysis.Geometry.Solid.3D.PART</precondition>
<precondition>Analysis.Fluids.Compressible.3D.TOTAL_PRESSURE</precondition>
<postcondition>Analysis.Structural.Static.3D.MAX_STRESS</postcondition>
              </conditions>
            </service>
            <service name="org.apdl.Analysis.Geometry.Solid.3D.Update.CAD2">
              <description>
                <ontology>Analysis</ontology>
                <QoS name="VariableCost">
                  <qValue>
                    <min>9.75</min>
                    <max>9.75</max>
                    <typical>9.75</typical>
                  </qValue>
                </QoS>
                <QoS name="ExecutionTime">
                  <qValue>
                    <min>9.0</min>
                    <max>39.0</max>
                    <typical>28.0</typical>
                  </qValue>
                </QoS>
                <QoS name="Reliability">
                  <qValue>
                    <typical>0.81</typical>
                  </qValue>
                </QoS>
                <QoS name="RSquared">
                  <qValue>
                    <min>1.0</min>
                    <max>1.0</max>
                    <typical>1.0</typical>
                  </qValue>
                </QoS>
              </description>
              <conditions>
```

```
<precondition>Analysis.Geometry.Solid.3D.MASTER_PARAMETERS</precondition>
                <postcondition>Analysis.Geometry.Solid.3D.PART</postcondition>
            </conditions>
        </service>
        <service name="org.apdl.Analysis.Meshing.Grid.3D.Mesh2">
            <description>
                <ontology>Analysis</ontology>
                <QoS name="VariableCost">
                    <qValue>
                        <min>0.75</min>
                        <max>0.75</max>
                        <typical>0.75</typical>
                    </qValue>
                </QoS>
                <QoS name="ExecutionTime">
                    <qValue>
                        <min>12.0</min>
                        <max>42.0</max>
                        <typical>27.0</typical>
                    </qValue>
                </QoS>
                <QoS name="Reliability">
                    <qValue>
                        <typical>0.85</typical>
                    </qValue>
                </QoS>
                <QoS name="RSquared">
                    <qValue>
                        <min>1.0</min>
                        <max>1.0</max>
                        <typical>1.0</typical>
                    </qValue>
                </QoS>
            </description>
            <conditions>
                <precondition>Analysis.Geometry.Solid.3D.PART</precondition>
                <postcondition>Analysis.Meshing.Grid.3D.MESH</postcondition>
            </conditions>
        </service>
        <service name="org.apdl.Analysis.Fluids.Compressible.3D.Fluids2">
            <description>
                <ontology>Analysis</ontology>
                <QoS name="VariableCost">
                    <qValue>
                        <min>2.75</min>
```

```
                        <max>2.75</max>
                        <typical>2.75</typical>
                     </qValue>
                  </QoS>
                  <QoS name="ExecutionTime">
                     <qValue>
                        <min>20.0</min>
                        <max>500.0</max>
                        <typical>50.0</typical>
                     </qValue>
                  </QoS>
                  <QoS name="Reliability">
                     <qValue>
                        <typical>0.97</typical>
                     </qValue>
                  </QoS>
                  <QoS name="RSquared">
                     <qValue>
                        <min>0.55</min>
                        <max>0.97</max>
                        <typical>0.94</typical>
                     </qValue>
                  </QoS>
               </description>
               <conditions>
                  <precondition>Analysis.Meshing.Grid.3D.MESH</precondition>
<postcondition>Analysis.Fluids.Compressible.3D.TOTAL_PRESSURE</postcondition>
               </conditions>
            </service>
            <service name="org.apdl.Analysis.Structural.Static.3D.Stress2">
               <description>
                  <ontology>Analysis</ontology>
                  <QoS name="VariableCost">
                     <qValue>
                        <min>4.75</min>
                        <max>4.75</max>
                        <typical>4.75</typical>
                     </qValue>
                  </QoS>
                  <QoS name="ExecutionTime">
                     <qValue>
                        <min>9.0</min>
                        <max>20.0</max>
                        <typical>15.0</typical>
                     </qValue>
                  </QoS>
```

```
                <QoS name="Reliability">
                  <qValue>
                    <typical>0.94</typical>
                  </qValue>
                </QoS>
                <QoS name="RSquared">
                  <qValue>
                    <min>0.50</min>
                    <max>0.98</max>
                    <typical>0.84</typical>
                  </qValue>
                </QoS>
              </description>
              <conditions>
                <precondition>Analysis.Geometry.Solid.3D.PART</precondition>
<precondition>Analysis.Fluids.Compressible.3D.TOTAL_PRESSURE</precondition>
<postcondition>Analysis.Structural.Static.3D.MAX_STRESS</postcondition>
              </conditions>
            </service>
            <service name="org.apdl.Analysis.Structural.Static.2D.StressCalculations">
              <description>
                <ontology>Analysis</ontology>
                <QoS name="VariableCost">
                  <qValue>
                    <min>0.01</min>
                    <max>0.01</max>
                    <typical>0.01</typical>
                  </qValue>
                </QoS>
                <QoS name="ExecutionTime">
                  <qValue>
                    <min>0.01</min>
                    <max>1.0</max>
                    <typical>0.5</typical>
                  </qValue>
                </QoS>
                <QoS name="Reliability">
                  <qValue>
                    <typical>1.0</typical>
                  </qValue>
                </QoS>
                <QoS name="RSquared">
                  <qValue>
                    <min>0.40</min>
                    <max>0.60</max>
                    <typical>0.55</typical>
```

```
                    </qValue>
                  </QoS>
                </description>
                <conditions>
<precondition>Analysis.Geometry.Solid.2D.MASTER_PARAMETERS</precondition>
<precondition>Analysis.Fluids.Compressible.2D.TOTAL_PRESSURE</precondition>
<postcondition>Analysis.Structural.Static.2D.MAX_STRESS</postcondition>
                </conditions>
              </service>
            </services>
```