Computer-Aided Design and Applications

http://www.cadanda.com

Applying Graph Theory and Autonomous Agents to Automate Product Development Processes

Jordan J. Cox[1], Greg Roach[2], John Daley[3] and Jeff Amelang[4]

[1]Brigham Young University, cox@byu.edu
[2] Brigham Young University – Idaho, roachg@byui.edu
[3] Phoenix Integration, jdalely@phoenix-int.com
[4] Brigham Young University, jeff.amelang@gmail.com

ABSTRACT

Graph theory has been used to model simple and complex systems of a wide variety of contexts. In this paper, it is used to represent product development workflows so that alternative options can be explored before actual deployment occurs. The paper presents the methodology using graph theory fundamentals. It then shows the application to product development workflows. The paper also introduces autonomous agent theory as a way to produce automatic configurations of workflow as connected graphs of agents. A secondary calculus is introduced that is used to score the configurations with respect to program objectives of cost, time to execute, etc. The paper shows several examples using an implementation based off of a commercial software solution.

Keywords: autonomous agents, product development processes, automation.
DOI: 10.3722/cadaps.2008.161-177

1. INTRODUCTION

This paper presents methods based on graph theory for creating network models of product development (PD) processes and for using these models as the architecture for agent-based systems to execute the product development process, calculate process metrics, and simulate process behavior. The network models are more than information networks, they represent all the tasks and methods required to achieve the PD process objective. Some of these tasks and methods include parametric CAD/CAE/CAM models as well as other process artifact models, human decision agents, etc. Because the network model is complete for the target process, the agent network achieves automation of the process. The agent network is created by instantiating agents to the leaf-nodes of the graph which is the lowest level of the defined ontology of the network model. Different network calculus models can also be assigned to the agent models to calculate various process metrics.

The method presented by this paper provides a flexible approach to representing and managing product development processes, supports reuse through the dynamic creation and execution of agent-based workflows, and provides a method for creating network models of global systems using web service strategies. Current automation techniques used in industry often require considerable effort to develop and maintain especially as new technologies, models, and team members are introduced. Some level of automation is required to manage the complexity of processes, deliverables, and teams. Effective management of the integration of people, tasks, deliverables, analytical tools and their interaction in a global product development process environment is essential for the successful development of highly engineered products.

2. BACKGROUND

Agent-based systems and the associated methods and techniques have been used in a variety of biological and non-biological contexts such as predator-prey models of deer herds, combat animation, micro-air vehicle coordination, robotics and other applications where overall system goals are met by independently acting agents. Lander explains that there is no clear definition of an agent [8]. However, a common, high level definition is that an agent is a computer system capable of independent action on behalf of a user [13]. In the context of this paper, an agent is any actor, human, computer-system or otherwise that is capable of independent action in pursuit of some overall system goal. Despite lacking a commonly accepted definition of what an agent is, implementations have been quite diverse and varied, spanning multiple disciplines. Several of these implementations have been in product development.

2.1 Literature Review

Karpowitz provides a broad literature review of agent system implementations in product development [6]. 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 [8].
- Multi-agent systems require minimal changes to existing tools and processes [5].
- Agent teams can be used in the conceptual design phase to find optimal configurations [4].
- A design-oriented model can be used with an agent system to create an automated product development system [1].
- 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 [5].

Existing agent-based systems for product development and process planning automation include PACT, SHARE, First-Link, Next-Link, Process-Link, and DIDE [12] [11].

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 [3].
- Agents can be used to build an application by selecting web service implementations that best match the quality criteria of the application [9].
- Ontologies and semantic web service descriptions can be used to dynamically discover potential workflows to meet system objectives [7].

Agents can be used to create dynamic workflow for simple design tasks [8].

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. What is lacking is a methodology for evaluating and optimizing larger processes composed of these services.

2.2 Product Development Processes

Product development is a complex process involving the coordination of many activities that transform customer and company inputs into fungible products and services. It is inherently an agent-based process with many human agents involved in the transformation of inputs to outputs. The human agents provide great flexibility and robustness in the process but they also introduce great variability and inconsistency. For this and other reasons, it is difficult to execute a product development process twice and get the same results. Many of the tasks in a product development process do not require a high level of flexibility and can be standardized and automated. Other tasks require a great level of flexibility and cannot be standardized. To address the needs for flexibility and standardization, a hybrid product development process is proposed that is composed of human agents and software agents. The new process is

constructed so that it can produce a given product and a family of derivative products customized around a predetermined envelope of product variations. The new product development process can therefore be used more than once, thus justifying the investment in capturing and automating it using agent methods.

Since the hybrid process involves human agents and software agents, each class of agent must be focused on accomplishing tasks that it is best suited to complete within the overall product development process. Software agents are able to perform repetitive tasks consistently and accurately. Human agents are able to make decisions and react to changing environments. To take advantage of these agent strengths, a hybrid process is created that consists of a workflow made up of software agents and human interaction agents. The workflow can then be executed to take the human agents through all the tasks, calculations, models, etc. needed to create all product deliverables for a specific member of a family of product derivatives.

The human agents operate this framework as though it were a product derivative generator. They are then able to explore a greater number of product variants, connect optimization engines to the generator, and generally explore product design space more readily. We refer to the creation of this product derivative generator as a PDG. The methods in this paper focus on decomposing an existing "virtual" product development process as though it existed in some real form. The decomposition allows the identification of the software agents and the creation of the workflows. However, these workflows are not static. Mapping agents are introduced that are not part of the product development process but organize process agents into run-time workflows and therefore allow for the workflow structure to be derived at execution based upon a library of available task agents.

This approach aligns with a web-services architecture, where a company might establish a registry of product development task agents that can be connected dynamically, based on dependencies, into a workflow so that a more flexible structure results. This allows exploration of a wider envelope of derivative products including possibilities for topological product optimization as well as the execution of alternative process paths to support dynamic process objectives. These methods have been successfully applied to existing processes to demonstrate feasibility and effectiveness of the strategy.

3. METHOD

The method of creating agent-based models of product development processes is accomplished by creating a network model of the desired process using graph theory and using it to identify all the necessary tasks of the "to-be" automated process. The tasks are the leaf nodes of the graph and form an architecture. Once the architecture is created, agents are identified which match each of the tasks of the architecture. These agents, therefore match one-to-one with the fundamental tasks of the process. The agents can then be reconfigured at run time using deterministic mapping to create an executable form of the process network. The method is achieved in three phases; first, creation of the process network architecture, second, defining a network calculus, and third, instantiation of the network model.

3.1 Creating the Process Network Architecture

Network models of processes have been created and used for many different purposes. Some of the most common network models are based upon causal networks formed by directed-acyclic graphs (DAG's) which connect events (nodes) with relationships of causality (edges) in a graphical structure. Probabilities of events occurring can then be calculated using Bayesian theory. Braha and Bar Yam [2] use graph theory to construct a network model of information flow in product development processes. These graphs can become rather complex and the models difficult to manage because of the complexity. Our method focuses on different levels of complexity associated with different ontological levels in the model to alleviate the difficulties with overwhelming complexity. The method is based upon graph theory and parallels some of the techniques used in Bayesian network models.

The creation of the process network model is based upon selecting an ontological classification scheme for the selected process and using the ontological classification scheme to decompose the process steps into the individual tasks that must be completed to accomplish the overall process objective. This is accomplished in three steps; first, a process ontology is selected, second, the elements of the ontology are delineated, and third, the network architecture is formed by deterministic mapping.

Selecting a process ontology involves choosing a hierarchical classification scheme that will encompass all pertinent elements of the process. It is an abstraction hierarchy. At the highest level, the overall process is represented and at the lowest level individual tasks are identified and represented. The number of levels in the hierarchy is determined by the

needs of the network model. For this paper, a process ontology which includes four levels will be used: an overall process level, a functional module level, a sub-function level and a task level. These four levels will provide enough resolution to demonstrate the method.

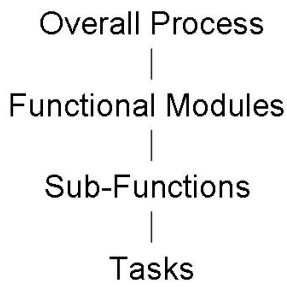

Fig. 1: A process ontology or abstraction hierarchy.

The second step is to delineate the elements of each level of the ontology. Essentially, the target process must be decomposed into elements at each level of the ontology. This is done by first considering the overall process to be an unknown function **F(ϕ)=Ω.** The inputs come from the customer and internal company knowledge and the outputs are the product deliverables. At the highest level, the overall process, the only element is the process function. The function **"F()"** is a composite function made up of a host of functional modules related to the process functions required to design and build the product. These could include defining a product configuration, making product predictions and engineering calculations, developing CAD models and drawings, technical reports, etc. These functional modules are elements of the second level of the ontology and can be organized into a schematic representation of input maps and output maps linked together by functions which convert inputs into outputs. Initially, the methods of converting inputs to outputs are not important; only the identification of the functional modules and their inputs and outputs.

To facilitate the development of the network model, graph theory is used for representation purposes. The process ontology at each level can be represented by separate graphs. This is done by representing the ontological level as a graph formed by connecting nodes or vertices, by arcs representing the inputs and outputs and their causality, see Figure 2.

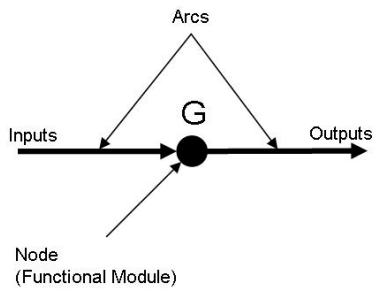

Fig. 2: A graph of a transformation of inputs to outputs.

The first level of the process ontology is therefore represented as the graph shown in Figure 3. This is of course almost trivial at this level. However, the lower level graphs are non-trivial.

Using graph theory, a graph will be denoted in the form:

$$G = (X, \Gamma) \tag{3.1}$$

where **X** is the set of vertices or nodes and **Γ** is the set of arcs connecting those vertices. Therefore, the top level representation of the process is:

$$G = (F, (\Omega | \Phi)) \tag{3.2}$$

where **F** is the single node representing the overall process functional module that converts the inputs **Φ** into the outputs **Ω**. Note that we use the same form as the probability function $P(\Omega | \Phi)$ to connote inputs Φ and outputs Ω. At this level of abstraction, there is no need to deterministically map the nodes into a network because of the simplicity of the graph.

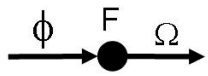

Fig. 3: Top level graph representation of the product development process.

The second level of abstraction has more elements. It includes a decomposition of the process into a set of all the functional modules that could make up the overall process. We use the same delineation as Hopkins and Pearl, i.e. we will use upper case letters (e.g. X, Y) to represent variables, and the lower-case correspondent (e.g. x, y) to represent a particular value of the variable. We will also use bold-face upper-case letters to denote sets of variables and bold-face lower-case letters to denote corresponding values of the sets.

The elements of the second level of abstraction are cast in the form of abstract function definitions **f(a)=b**, where the form of **f** is not necessarily known. These abstract functions do not yet need to be associated with a specific process. Therefore, this level of the ontological abstraction is specific to product development processes because of the selection of the functional modules, but is indistinguishable from one product development process to another. Some decomposition of the process into functional modules must be selected. For example, assume a decomposition of a product development process into the following functional modules:

D(C)=M1; Transformation of customer inputs (C) into product parameters (M1)
R(K)=M2; Transformation of company rules (K) into company parameters (M2)
M=M1UM2; Integration of the product parameters into a master set (M)
P(M)=B; Transformation of product parameters into product behavior predictions
G(M)=A; Transformation of parameters into artifacts (drawings, reports, etc.)
A=A1UA2UA3; Integration of artifacts into a common set A
I(A1)=T; Transformation of artifacts into test results
E(A3)=V; Transformation of artifacts into vaultable information
S(A2)=U; Transformation of artifacts into customer deliverables.

An additional node Q is included to account for iterative activities that occur to determine appropriate values for the product function. These functional modules will now form the partitions of the original sets **X** and **Γ**.The set **Γ** for a product development process, at the second level of abstraction, then must include initial inputs and outputs of the process as well as any intermediate inputs and outputs of the functional modules. **X**, at the second level of abstraction, is defined as all the nodes that represent functional modules that change inputs into outputs. The partition of **X** is defined as a set of sets of nodes representing the functional modules of the process. Script letters are used to differentiate between nodes and arcs.

X = $\{\mathcal{D}, \mathcal{R}, \mathcal{P}, \mathcal{G}, \mathcal{I}, \mathcal{E}, \mathcal{S}\}$ where,

$\mathcal{D}$ is the set of nodes transforming customer inputs into product parameters,
$\mathcal{R}$ is the set of nodes transforming company rules into product parameters,
$\mathcal{P}$ is the set of nodes transforming product configuration into predictions,
$\mathcal{G}$ is the set of nodes associated with creating product artifacts,

$\mathcal{J}$ is the set of nodes associated with test and prototyping,
$\mathcal{E}$ is the set of nodes associated with data vaulting
$\mathcal{S}$ is the set of nodes associated with product finalization and delivery,
$\mathcal{Q}$ is the set of nodes associated with design iteration.

Next, $\mathbf{\Gamma}$ is partitioned into a family of sets corresponding to inputs and outputs of the major functional parts of the product development process defined previously.

$\mathbf{\Gamma}$ = {**C, M, K, B, A, T, V, U**} where,
C is the set of customer requirements,
M is the set of master parameters of the product configuration,
m1 is a subset of **M** associated with customer inputs,
m2 is a subset of **M** associated with company knowledge,
The sets **m1** and **m2** combine to form the set **M**,
K is the set of inputs defined by company knowledge,
B is the set of predicted product performance metrics,
T is the set of test outputs,
V is the set of information vaulted by the company, and
A1-3 are the sets of product artifacts created by the process
U is the set of product deliverables sent to the customer.

These partitions of the sets **X** and $\mathbf{\Gamma}$ are somewhat arbitrary, meaning that a different decomposition could be used without any impact on the method. The partitions therefore are determined by what decomposition best provides elements that are of interest in the model. Functional modules are identified and all product development functions are elements of one of the modules (nodes) or inputs and outputs (arcs). Once these partitions are identified, the overall graph must be created by backwards mapping. In essence, the graph is constructed by identifying the outputs of the process and then connecting them with the associated nodes (functional models) whose outputs are the outputs of the process. Next, the inputs to these nodes or modules are represented by arcs and a new set of nodes are connected and so forth until all nodes and arcs are represented. The partial graph of the second level of abstraction of the product development process is represented as shown in Figure 4.

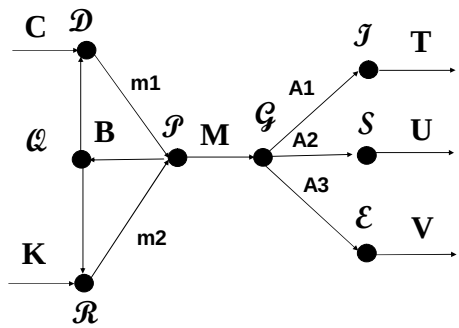

Fig. 4: Second level graph of a product development process.

This graph is related to the first level abstraction as would be expected. The arcs **C** and **K** are partitions of the input arc **Φ**. The arcs **T**, **U**, and **V** are partitions of the arc **Ω.** All other nodes and arcs are partitions of the function **F** as can be seen in Figure 5.

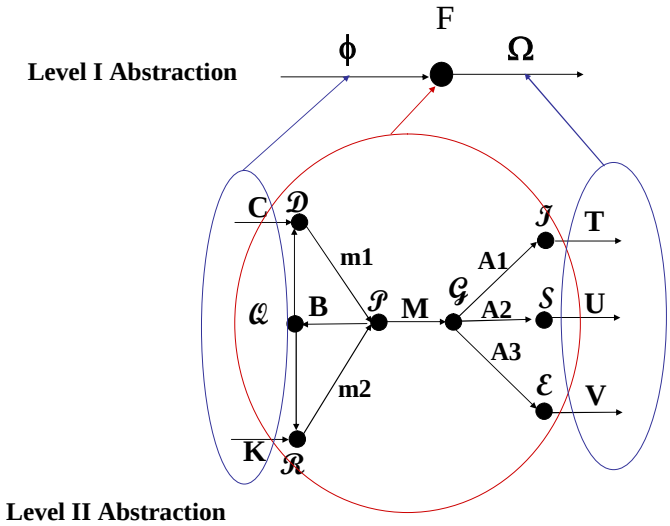

Fig. 5: Relationships between the first level of abstraction graph and the second level of abstraction graph.

The graph **F=(X, Γ)** at the second level of abstraction is now represented with the sets **X** being partitioned into the nodes: **D, P, Q, R, G, I, S, E**, and the set **Γ** being partitioned into the arcs: **C, K, m1, m2, B, M, A1, A2, A3, T, U, V**. Additionally, the sets **Φ** and **Ω** are associated with the arcs: **C, K**, and **T, U, V**, respectively.

The third and fourth levels of the ontological abstraction are constructed in much the same way as the second level graph. However, at the third and fourth levels, the graphs become process specific. A specific product development process must be selected and the elements of the third level representation come from the chosen process. For the third level, each node in the second level is partitioned into associated sub-functions. This means that for example, all the necessary sub-function modules needed to accomplish the specific functions of a chosen process are identified. Once all the sub functions are identified they are backwards mapped into the graph. The fourth level graph is created by partitioning the third level nodes into their associated tasks and then backwards mapping to form the fourth level graph. It should be obvious that the graphs of these product development process levels of abstraction can become rather difficult to represent and manage. The use of software tools becomes necessary. To illustrate these third and fourth level graphs, we present the product development process for a digital thermometer as shown in Figure 6.

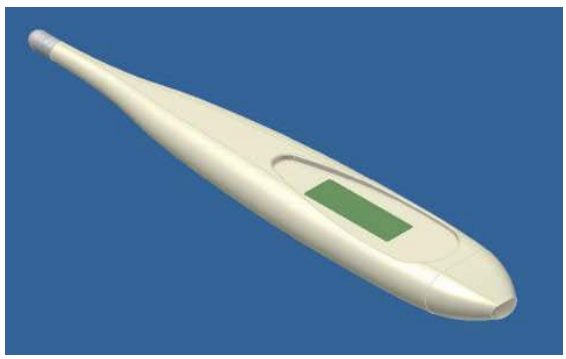

Fig. 6: Digital thermometer.

Two different versions of the graph are shown. An arc-centric graph is shown in Figure 7 first, to delineate the inputs and outputs of the functional modules identified in the level 2 ontology. The ovals in the figure represent nodes from the level two ontology.

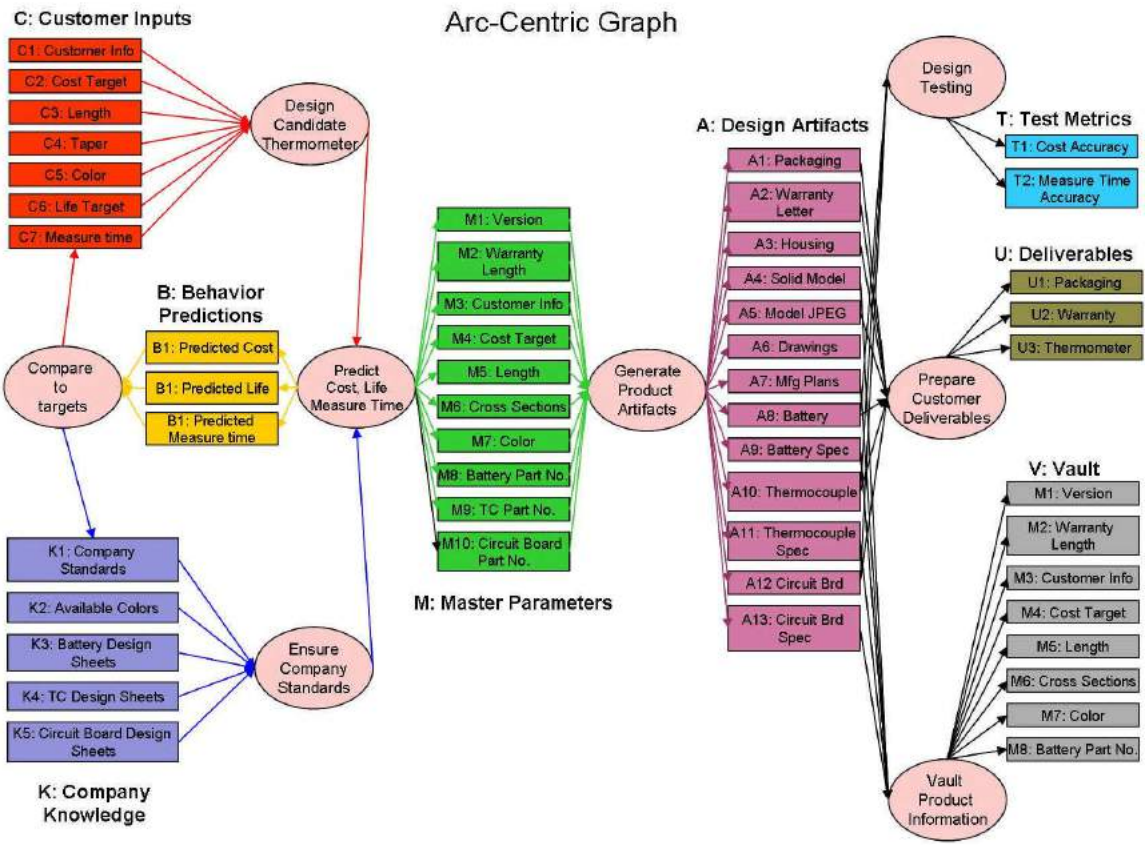

Fig. 7: Arc-centric Graph of the third level of a simple product development process.

Next, a node-centric version of the graph is shown which delineates the sub-functional modules of the third level ontology. These are shown in Figure 8.

The pink nodes of Figure 7 have been partitioned into the multi-colored nodes in Figure 8.

The final level or fourth level ontology represents the lowest level or tasks within the product development process. Each sub-function node shown in Figure 8 must be defined in terms of the tasks needed to complete the sub-function. An example of a fourth level node or sub-function is shown in Figure 9.

Enumeration of the third and fourth level graphs often comes from knowledge of former product development processes as they relate to the target process. There are common elements such as the final product, warranty manuals, vaulted drawings and documents, etc. and there are desired elements such as test validations, product performance predictions, etc. All desired final outputs of the process must be specified. This is done by specifying each of the sets:

The sets could be defined for a given process as:
B={**b**| **b**is an element of (Max stress, life, cost)}
T={**t**| **t**is an element of (stress measurement, life cycles to failure, actual cost)}
U={**u**| **u**is an element of (Final product, Warranty, Packaging)}
V={**v**| **v**is an element of (vaulted drawings, prediction reports)}

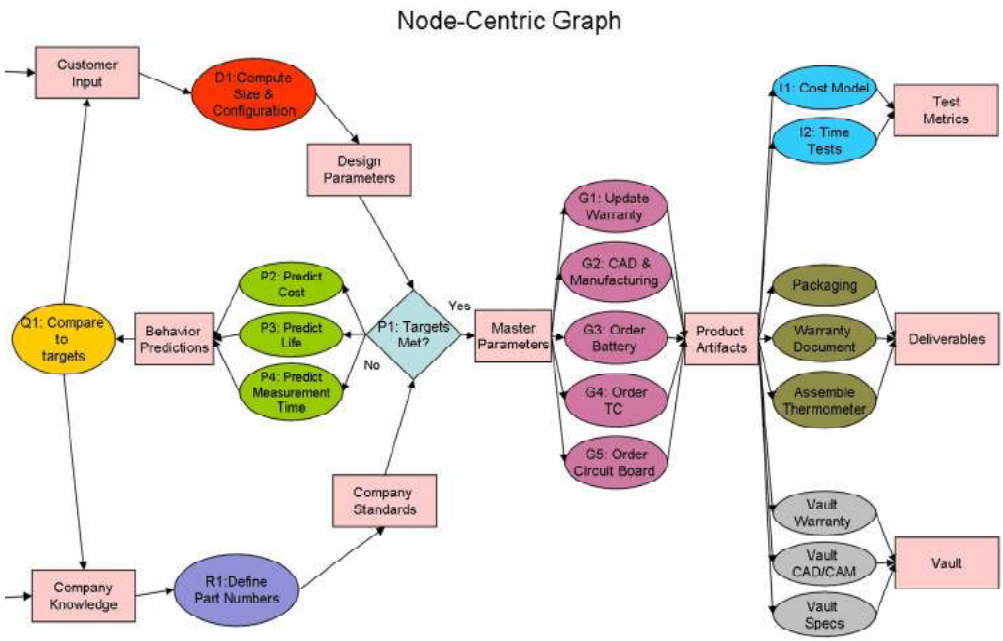

Fig. 8: Node-Centric graph of the third level ontology.

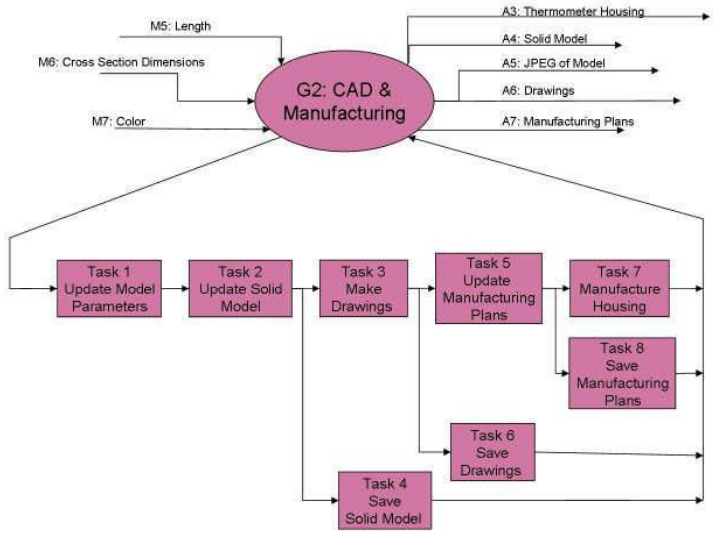

Fig. 9: Tasks in the lowest level ontology which constitute one sub-function in the third level ontology.

Notice that enumeration of the members of the node sets actually begins to structure the process graph for a specific product family. It is specific to a given process. At this point, not all of the elements are always known. However, the

better the enumeration, the more complete the graph will be. It is possible to update the elements of these sets as the graph is refined and then the graph can be updated.

3.2 Defining a Network Calculus

Once the network has been created as a fourth level graph made up of all the tasks associated with the process, a network calculus (system of calculating process metrics) must be defined over the graph, so that calculations in the form of process behavior and predictions can be made. There are many different forms of the network calculus depending on what the model will be used for. For example, inherently, the network was created to represent the product development process which has a calculus associated with creating the product deliverables. This PD calculus can be defined based upon delineating the form of each of the functions in the lowest level of the abstraction. This means that each task is defined in terms of how it transforms its inputs into its outputs. For example, if a stress prediction is to be calculated, the actual equations must be defined. Once all the tasks are delineated, an overall network calculation can be made. A Bayesian network calculus can be used to represent the probability that PD events will happen. Other types of calculus can be applied to these ontological graphs to estimate cost, timing, quality, etc.

3.2.1 The Product Development Network Calculus

The original intent of the network model is to obtain an "executable" network that would represent the product development process. Once the individual tasks for the process are identified, the inputs can be mapped to the outputs using the appropriate engineering functions. Product parameters can be converted to solid geometric models using solid modeling equations, stress and fluid flow can be calculated from product parameters as well using field equations and constituent relationships, and artifacts can be converted into test results using test procedures. Implementing all of these tasks creates an "executable" network calculus that can be calculated across the network.

3.2.2 A Bayesian Network Calculus

Probabilities of events occurring in the network can be calculated once a Bayesian calculus is defined on the graph. This allows the calculation of probabilities throughout the network based on causality defined by the directed arcs of the graph. Nonlinear relationships require a more sophisticated approach presented by Pearl [1995a, 2000].

3.2.3 Other Types of Calculus

Often, measures of effectiveness of the network are of value in determining network optimality. For example, measures of cost, time and quality of the engineering network calculus can be of particular interest. In order to implement these types of calculus, the calculus is implemented at a higher ontological level. Cost, time and quality are defined for each of the tasks at the sub functional level. The total network values are then determined using appropriate summations based upon desired accumulation values. For example, total cost or time would be a linear summation over the entire network, where as total quality might be a product over the network of quality measures or uncertainty values.

3.3 Instantiation of the Network Model

The final phase of developing a network model is the instantiation of the model with agents. The concept is that the process is represented by the network models, but the execution must be accomplished by agents. These agents can be software or human agents. The model calculus is implemented in the agents. As the agents execute the process, the network metrics can be calculated using the defined calculus.

Selection of agents must be done based upon possible execution capabilities of the organization. Software agents can be used if they exist and are validated. Otherwise, human agents must be used to execute the workflow. The most straight forward method of instantiation is to define a separate agent for each node in the network model. This can be done for each of the ontological levels. At the lowest level, each task has an associated agent. An example using the partial graph of Figure 9 is shown in Figure 10.

The graph model now becomes an architecture for managing the agents of the network. For example, there may be multiple agents defined to accomplish each defined task associated with different methods used to accomplish the tasks. This allows for different methods to be embodied by the agents. For example, if a task is defined to calculate stress values for a given part of the product, an automated software agent can be defined that uses closed form constitutive equations. Another software agent can be defined that uses finite element methods, and a third human agent can be defined that uses empirical methods. If all other tasks have single agents associated with them, then three

different instantiations of the overall process can occur; one with each of the three redundant agents. The overall cost, time and quality of the network will be different for each instantiation.

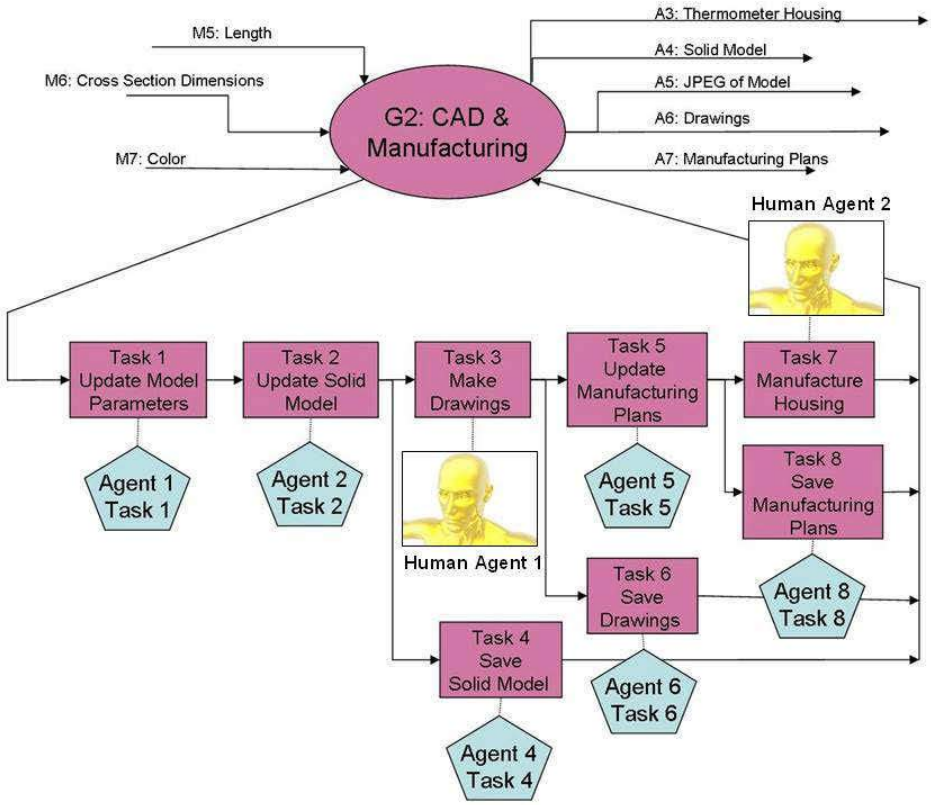

Fig. 10: Simple agent instantiation.

The complete model can now be created through implementation in an agent environment. For the purposes of this paper, iSIGHT-FD was modified to provide such an environment. iSIGHT-FD is a commercial software solution offered by Engineous Software Inc. It is used primarily as a workflow integration environment but has been modified for this work. Using the graph models, a list of agents were created for the thermometer problem. Each agent must contain its own procedural algorithms and methods to accomplish its assigned task. These agents were then created in iSIGHT-FD as stand-alone modules. The modules are then published to a library agent which is an agent that simply maintains a registry of all agents and their inputs and output requirements.

A dynamic workflow mapping agent is then invoked to assemble a workflow for a given request. The mapping agent searches the registry for an agent whose output matches the request and places it at the end of the process. It then takes that agent's inputs and uses them as the request for a second search in the registry. This continues until all requests are satisfied or there are no more agents available for use in the workflow. The dynamic workflow mapping agent is critical since it delineates all the possible workflows. It is essentially a "backwards" mapping or deterministic mapping technique that is used by the mapping agent to determine workflows. The resulting workflow architecture for the example is shown in Figure 11.

Each of the "scrolls" in Figure 11 represents a different agent. The arrows represent the input-output relationships between the agents. The overlapping of agent titles occurs because of the dynamic instantiation and also because the software is not set up to automatically space the graph for agent titles. This workflow is executable since each of the agents contains its own procedural methods for completing its own task.

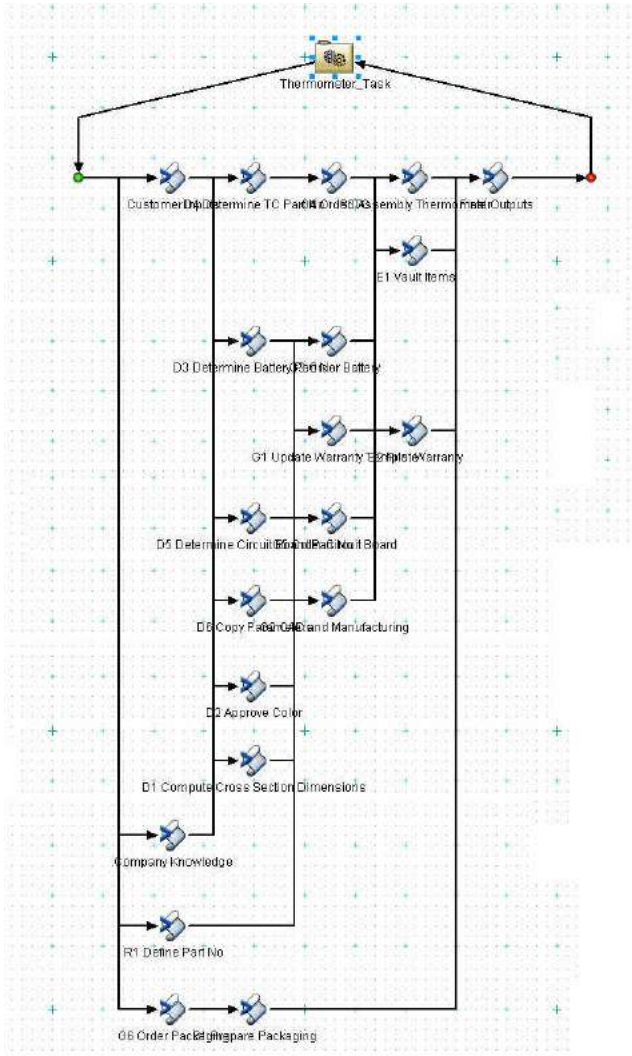

Fig. 11: Dynamic workflow for the thermometer design process.

4. DYNAMIC NETWORK MODELS

Often, changes in the network model occur during the execution of the product development process. Seldom is a network model static throughout the execution. Therefore, a dynamic form of the network is necessary to deal with the changes. One example of this is when there are multiple agents that can accomplish the same task. This provides for many different instantiations of the network model. Design fidelity can be controlled using different agents. Preliminary design may be executed by instantiating the network model using 1-D or low fidelity agents. Detailed design could then follow by instantiating the network with higher fidelity agents.

For example, consider a simple partial graph of a product development process for designing aircraft engines made up of the four tasks shown in Figure 12. The partial graph represents four tasks in the sub-function of designing a radial compressor. The four ontological levels are shown in Figure 13. A simple one-to-one instantiation of agents is used initially. The figure also shows a software implementation where the process now is composed of the active instantiated agents.

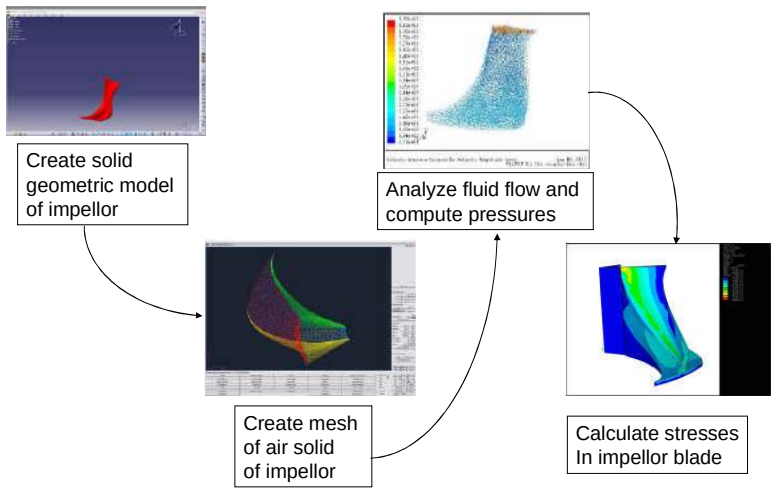

Fig. 12: Four tasks in an aircraft engine design process.

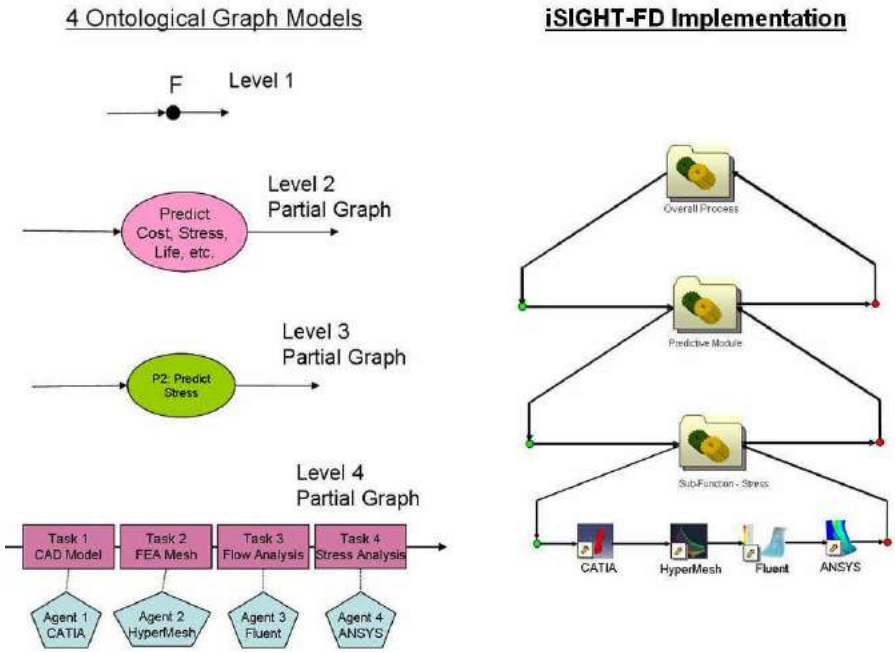

Fig. 13: Ontological levels and software implementation.

A dynamic network model is achieved through the use of an agent for doing the deterministic or "backwards" mapping of the network. The method used, as explained earlier, is to delineate the nodes at a given ontological level of the network, define a list of agents that exactly match to the nodes and then to define a dynamic workflow network of these agents by forming a query for the output of the overall process. The agent that produces this final output is instantiated into the agent environment and its inputs form the next level of queries. Agents are then found which

satisfy the new queries and this process continues until all agents are instantiated into the workflow. Special cases of cyclic queries, redundant outputs or inputs, etc. must be handled. The result is a workflow of the lowest ontological level of the network made up of the instantiated agents. Figure 13 shows a software implementation using iSIGHT-FD. The dynamic process is represented by the instantiated agents of the lowest ontological level. It has a unique graph topology, although trivially simple in this case. When this method of backwards mapping is used, whenever a change occurs in agents, a new network model can be identified. This allows for dynamic network identification, topological studies and just-in-time network models. Situations which require these methods include changes in agent availability, introduction of new technologies, redundant agents, and changes to the process topology, and reaction to process failures.

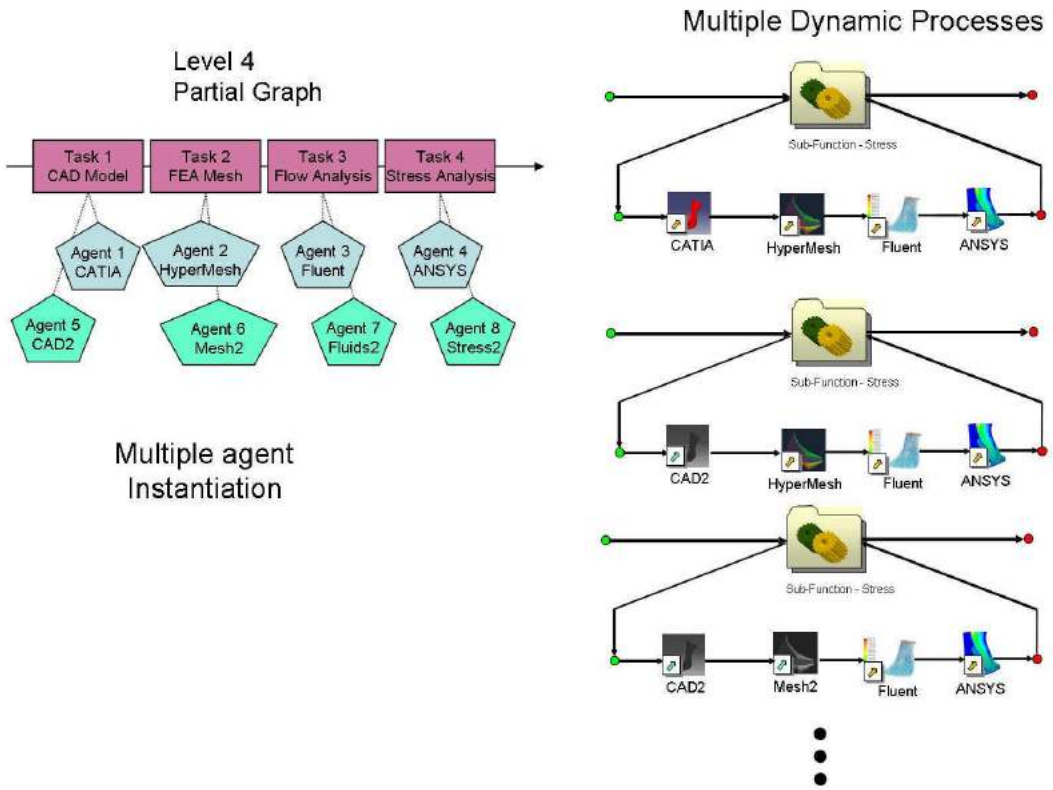

Fig. 14: Multiple processes resulting from dynamically mapping multiple agents.

This also allows for comparisons of different agent instantiations to the network model. Optimality studies can be performed to determine best case scenarios for execution of the process. Figure 14 shows this same process but a redundant instantiation of agents. Each task has two agents that can perform it. When the dynamic agent mapping occurs, multiple process instantiations result as indicated by the iSIGHT-FD processes composed of different combinations of agents. With two agents for each task, there are 16 process combinations that can calculate the desired stresses. A secondary calculus is defined to calculate efficiencies such as cost, time and quality. The calculus is defined within each agent. Each of the 16 process combinations will therefore have a unique efficiency total. The processes can then be chosen based upon these efficiency totals. Table 1 shows the values used in the efficiency calculus.

Figure 15 then shows an optimal ranking of the 16 processes based upon cost being the most important. Process one, in this case, is the optimal for cost. Figure 16 shows the optimal for reliability and process 6 is the optimal process when reliability is most important. Figure 17 shows another weighted optimization where speed and reliability are important and in this case process 4 is the optimal process.

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

Tab. 1: A secondary calculus defined for each agent.

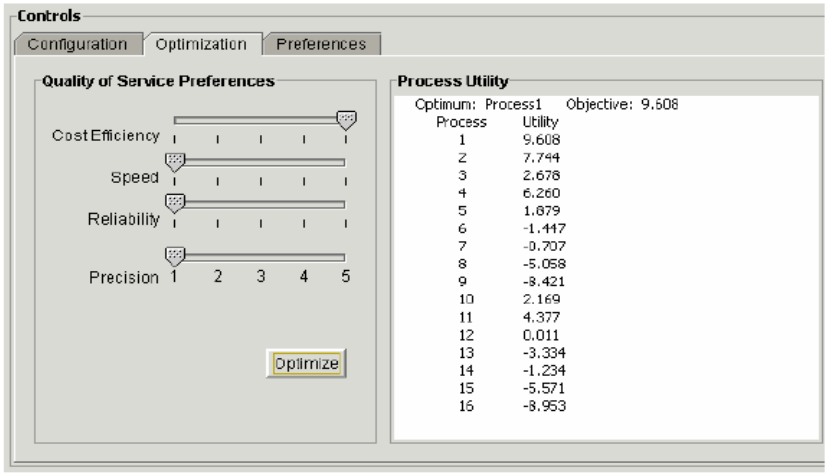

Fig. 15: Comparison of the 16 processes based upon total cost.

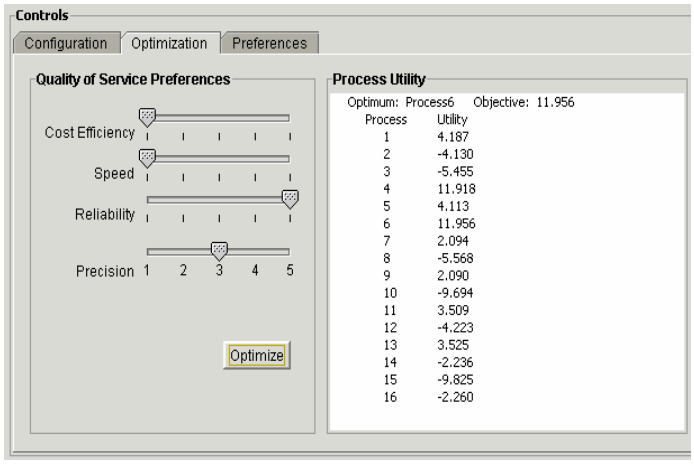

Fig. 16: Comparison of processes based upon reliability being the most important aspect.

The secondary calculi posed upon the agent instantiations provide a method for doing weighted comparisons of the different process combinations.

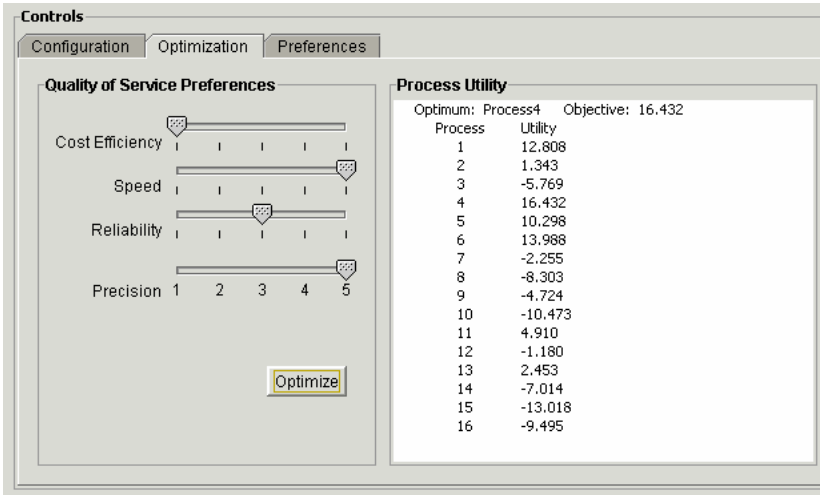

Fig. 17: Comparison based upon speed and reliability.

5. SUMMARY AND CONCLUSIONS

Automation of the product development process is achieved by first creating a graph model of the process. This involves defining ontological levels to manage the complexity of the graph models, delineation of the nodes of each ontological level and then deterministic mapping to complete the graph and satisfy all dependencies. Once the graph model is complete, a list of agents can be created from the list of nodes in the graph. Procedural methods for each agent must be defined so that the agent will be able to complete its assigned task. A dynamic network workflow model is then created in an autonomous agent environment by creating a registry of all available agents and a mapping agent that can satisfy queries by using agents in the registry. When all the available and appropriate agents have been placed in the workflow model, the process can be executed since all agents have their own procedural methods. Reconfiguration of the workflow model can occur if changes occur to agents. Virtual product development studies can be accomplished by creating all possible agents that could accomplish all the tasks in the product development process and then allowing the dynamic mapping agent to map out all possible workflows. These workflows can then be evaluated through the use of a secondary calculus of process metrics measured at the agent level.

The following conclusions can be made from the methods presented:

1. It is possible to represent product development processes using graph theory,
2. Complexity can be managed through the use of ontological levels and multiple graph models,
3. The use of autonomous agents can convert the graph models into dynamic workflow models.
4. In an autonomous agent environment, individual agents can be created with their own procedural methods to accomplish their specifically assigned task,
5. The process model can be executed to produce the desired product deliverables,
6. Multiple process models can be explored in a virtual product development study when all forms of agents are created for each task in the process and the mapping agent identifies all possible workflows,
7. The multiple process models can be evaluated and compared by applying a secondary calculus on the agents and across the network.

These methods have been used to model product development processes in industry for the purposes of automating a specific process. For example, an aircraft engine manufacturer has a preliminary design code that is used to explore new engine configurations. The current implementation is very fragile with respect to changes in specific tasks in the workflow. These methods are being studied as a method to capture task specifics in agent libraries that allow dynamic configuration of workflows and thus greater flexibility. The greatest challenge currently is finding commercial software solutions that can handle complex cases. Future work will focus on defining software tools and environments for commercial cases.

6. REFERENCES

[1] Birmingham, W. P.; Darr, T. P.: Automated Design for Concurrent Engineering, IEEE Expert, 9(5), 1994, 35-42.

[2] Braha, D.; Bar, Y. Y.: Information flow structure in large-scale product development organizational networks, Journal of Information Technology, 19, 2004, 244-253.

[3] Buhler, P.; Vidal, J. N.: Adaptive Workflow = Web Services + Agents, Proceedings of the International Conference on Web Services, Las Vegas, NV, 131-17, CSREA Press, 2003.

[4] Campbell, M. I.; Cagan, J.; Kotovsky, K.: A-Design: An Agent-based Approach to Conceptual Design in a Dynamic Environment, Research in Engineering Design, 11, 1999, 172–192.

[5] Feng, S. C.: Preliminary Design and Manufacturing Planning Integration Using Web-based Intelligent Agents, Journal of Intelligent Manufacturing, 16, 2005, 423-437.

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

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

[8] Lander, S. E.: Issues in Multiagent Design Systems, IEEE Expert 12(2), 1997, 18-26.

[9] Maximilien, E. M.; Singh, M. P.: 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.

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

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

[12] Sun, J.; Zhang, Y. F.; Nee, A. Y. C: A Distributed Multi-agent Environment for Product Design and Manufacturing Planning, International Journal of Production Research, 39(4), 2001, 625-645.

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

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