Computer-Aided Design and Applications
© 2009 CAD Solutions, LLC
http://www.cadanda.com

A Product Development Scenario for Knowledge Capture and Reuse

Joshua Bryson[1] Jordan J. Cox[2] and John T. Carson[3]

[1]Brigham Young University, bryson.joshua@gmail.com
[2]Brigham Young University, cox@byu.edu
[3]Lockheed Martin, john.t.carson@lmco.com

ABSTRACT

This paper focuses on proposing a product development scenario that embodies concepts of ontological knowledge structures that provide mechanisms to capture product and process knowledge during the execution of a development program. The ontological elements of this system are identified and defined in a storyboard that demonstrates the knowledge capture and reuse of the scenario. The purpose of this paper is to present a strategy that can be implemented in actual product development practice to capture and reuse product and process knowledge. This strategy involves capturing product knowledge in the form of requirements, artifacts such as CAD models, CAE models, process definitions, etc., in the context of a product development process template, so that the knowledge is easy to capture, reusable and transferable.

Keywords: product development, ontology, design editor, knowledge management.
DOI: 10.3722/cadaps.2009.207-218

1. INTRODUCTION

Most product development processes (PDP's) document the process as an afterthought. Usually information regarding how and why decisions were made is lost and only the end results are captured. Indeed, there are three main problems with knowledge capture and knowledge management (KM). The first problem is that knowledge capture often fails because it is time-consuming to implement and often is not incorporated into the engineering process or company organizational structures [10],[5]. The second problem is that captured knowledge is often unusable [3],[5]. Finally the third reason is that KM isn't just a horizontal problem (passing knowledge between current employees) but also extends vertically (passing knowledge between generations of employees) [8].

To address the first problem, several different knowledge capture solutions have been presented. Most of these solutions are web based and capture knowledge independent of the actual PDP. Udeaja et al. [14] describe a web-based solution that allows participants of a PDP to store and access knowledge gained during the PDP. Ajit et al. [1] provide a solution that simplifies the knowledge capture system by eliminating the need to pass knowledge to be captured to a knowledge engineer. Instead, they set up a design editor that gives the ability to capture and maintain knowledge to the domain experts. Baxter et al. [2] provides a combined process/knowledge based solution that "enables product data to be linked to the non-geometrical information through the process model." These solutions fall short because none of them are complete solutions; they focus on individual parts of the product development process but do not include the entire process from requirements to deployment.

To assist in the second problem, (i.e. knowledge usability) several ontological methodologies have been suggested. These tend to use knowledge annotations to provide more context for the captured knowledge. However few, if any, propose using the existing formats for the knowledge that is used to create it. For example, Li and Ramani [11] suggest that an increase in usability may be gained through placing design artifacts into a formal design ontology. As design artifacts are stored they are parsed for key words and placed into the correct part of the ontology. Each ontology part consists of sub-ontologies and the artifacts may be placed further in the overall ontology, yielding a more formal description of what that artifact really represents. Though not providing parsing and artifact storage methodologies, numerous other ontologies have been suggested to aid in artifact classification. Bohm et al. [4] describe an artifact centric data structure, or ontology, that consists of seven main classes or data types: artifact, function, failure, physical, performance, sensory and media. Kitamura et al. [9] suggest using a function centric ontology as a way to represent functionality of engineering artifacts in documents. Nanda et al. [12] use an ontology structure to aid in product family development. They do this by developing the ontology postproduction and then assigning finished artifacts and artifact pieces into the ontology. Sim and Duffy [13] developed their ontology by breaking the design process into three main design activities. These design activities are then further reduced into sub-activities. In addition to using ontologies to better represent, store and access design artifacts and information studies have been done to determine how to best represent that information. Hwang and Salvendy [7] studied the effects ontology display has on the usability of captured knowledge and suggest several ideas on best ontological representation (textual or graphical) depending on the situation. Although these solutions provide increased usability of stored artifacts the methods themselves are not always context specific, which hinders usability.

Another way of improving the knowledge usability problem is through addressing the requirements traceability problem. The ability to understand how a requirement changes during the product development process provides key information for knowledge reuse. Gotel and Finkenstein [6] investigate the sources of the requirements traceability problem and show that improvements can be made by better accessibility to sources of requirement changes.

To address the final problem, that of passing knowledge vertically, from generation to generation, Kalkan [8] suggests that this process is composed of seven stages. Each of these stages represents a vital part of the process of passing knowledge vertically. Although going through these stages would help vertical knowledge transfer, Kalkan [8] offers no suggestions as to how to properly implement his findings. The authors feel that if knowledge is captured in the context of the PDP so that it can be re-executed, this provides the greatest possibility for transferring knowledge.

To properly address these issues any solution needs to include these three parts: first, it should create a PDP tool that executes a PDP and captures the knowledge at the same time; second, it should be able to store all knowledge in its original form and context so that it can be effectively reused; finally, it should insure that the knowledge can be re-executed in the PDP tool so that it can be understood, reused and modified and therefore knowledge is passed horizontally and vertically.

Although the above approaches to the PDP problems help address and may even solve some aspects of the problems associated with KM none of these solutions is complete. A complete solution should capture knowledge and make it immediately reusable from the beginning of the PDP through the deployment and life of the product. Furthermore, it should capture knowledge automatically during the PDP. A proposed solution to the above issues is a product development tool that automatically captures knowledge during the execution of the PDP. In this paper, we propose to construct a solution using four knowledge tools, a product definition editor, a process definition editor, an agent editor and a deployment editor, to aid in developing a product and also to aid in implementing a deployable process once a finished design is reached. All activities and knowledge during the PDP are also captured the context of the process.

Each of these editors provides access to relevant past knowledge and information, such as the experience domain of the company for the given requirement, agent, deployment method, etc. Additionally, each editor will capture the design process as the designer interacts with the editor. This will capture most of the required knowledge to make knowledge reuse possible.

In addition to the editors, four knowledge engines will be used to facilitate storage and use of captured knowledge. A requirements engine tracks, stores, and allows searching of past and current product requirements. A configuration engine allows the designer to access past designs in terms of artifacts such as CAD models as well as process definitions. An experience domain engine gives access to knowledge gained by past experiences. A Mapping and optimization engine allows the designer to prepare a deployment plan for the product program and optimize the deployment *a priori*.

2. ELEMENTS OF THE PRODUCT DEVELOPMENT SCENARIO

The proposed solution involves defining web tools that help in the execution of a PDP. Knowledge capture and management is built under these tools in such a way as to automatically capture knowledge in the context in which it was created during the PDP. The proposed strategy is to break the PDP into its four basic phases and define the tools in such a way that the output of one tool is the input of the next. The first phase takes system requirements and divides those requirements into sub-system requirements and then into sub-subsystem requirements, etc., eventually leading to the requirements of individual components. Once component requirements have been defined solutions may be found and the system may be configured. Once the system is configured phase two starts. Phase two takes the system configuration and defines the processes needed to produce the configuration. This phase ends when all processes that are required to produce product deliverables are defined. The third phase defines what actors, (e.g. in-house vs. subcontractors) will do which processes defined in phase two. The last phase defines PDP programs that combine all processes into a deployable program plan.

The product development scenario is composed of a series of frameworks that will eventually be embodied in software tools. Underneath these frameworks are four knowledge engines that consist of data structures, repositories and logic engines. The frameworks are presented along with storyboards of the software tools that represent them. The four knowledge engines are then presented.

2.1 Product Definition Editor

This element provides a framework to track and aid the process of defining and decomposing high-level system requirements into subsystem and component requirements and then to convert them into the system configuration.

2.2 Process Definition Editor

This element provides a framework to track and aid the process of converting system requirements and configuration into an executable product development process that will produce the product deliverables.

2.3 Agent Editor

This element provides a framework for defining and managing agents that will execute the tasks of the product development process. Agents in this scenario are defined as actors assigned to tasks. Thus, this framework provides a repository for storing agent performance information and identifying metrics for monitoring their future performance on a specific product development program.

2.4 Deployment Editor

This element is the final framework for defining all the knowledge necessary to launch a product development deployment. Once the system requirements, configuration, process requirements, configuration, and agents are de-fined, an ontological logic engine can define all possible combinations of agents into a set of deployments. The metrics associated with the agents can then be accumulated across the deployments to determine scores for each deployment. The deployments can then be ranked to determine which one will be optimal.

2.5 Knowledge Engines

There are four knowledge engines that support the frameworks presented in the former section. These engines rep-resent knowledge structures and associated repositories that are used for storing the knowledge generated during a product development program. The engines also provide logic for analyzing the associated knowledge.

2.5.1 Requirements Knowledge Engine
This engine defines a hierarchical data structure for defining and storing knowledge in the form of requirements for product systems and processes. The engine also provides search capabilities.

2.5.2 Configuration Knowledge Engine
This engine provides a tree-based data structure for defining product configurations and process workflows. It also provides a multi-level representation so that configurations can be defined conceptually through sketches as well as fully defined through CAD models and drawings. This multi-resolution capability is also used to support process representations.

2.5.3 Experience Domain Engine
This engine captures knowledge gained over multiple product development programs in the form of trade-off plots. Each system, subsystem and component will have associated with it a series of trade-off plots. These plots can be 2-d or 3-d. These plots represent design space and usually, they will have some indication of regions within the design space that are preferred or where certain technologies or product solutions are used.

2.5.4 Mapping and Optimization Engine
This engine takes definitions of agents and process configurations and uses a backwards mapping or dependency mapping technique to define all possible combinations of possible deployments for a product development pro-gram. Once deployments are identified, secondary calculi must be defined across the graph models of the deployments. These secondary calculi provide methods for scoring each deployment based upon program goals of cost, quality, and time as well as secondary issues such as global geographic issues. The engine also provides optimization capability so that the deployments can be ranked or selected based on program criteria.

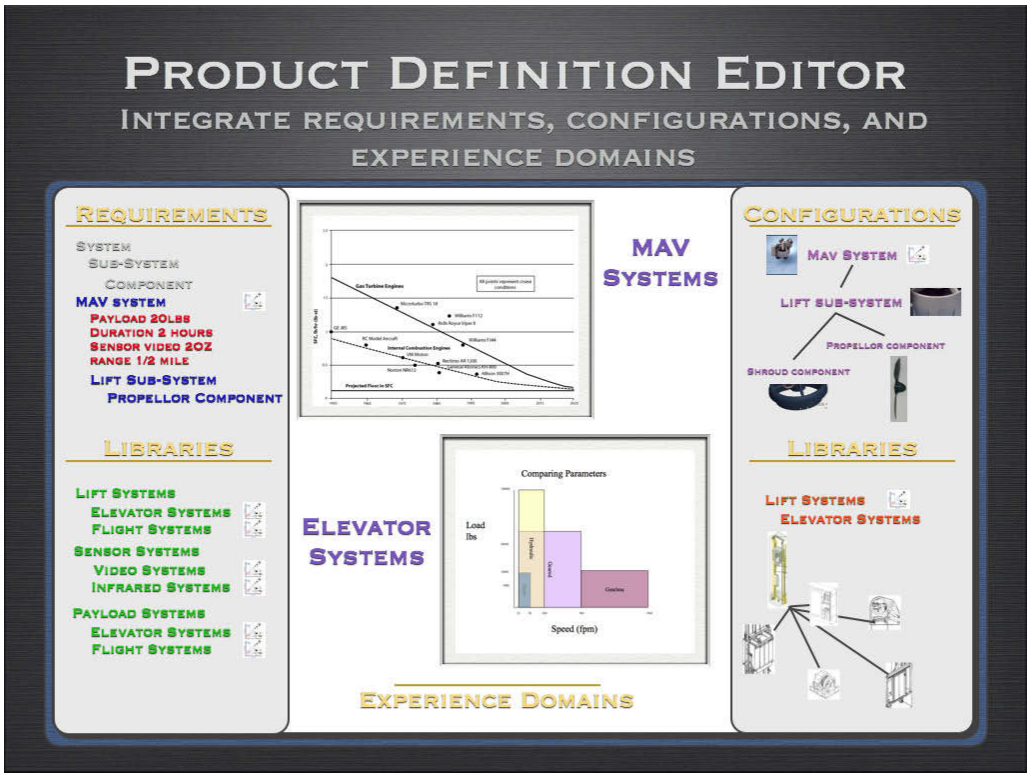
3. PRODUCT DEFINITION EDITOR
This element provides a framework to track and aid the process of defining and decomposing high-level system requirements into subsystem and component requirements and then to convert them into the system configuration.

3.1 Product Representation
Product representation is made up of two parts: requirements and configuration. The definition of system requirements and then the decomposition of these system requirements into subsystem requirements and eventually component requirements represent a significant part of the product representation. The activity of defining requirements is usually associated with preliminary design or system level design, which occurs during the initial stages of a product development program. However, requirements change throughout a product development program and therefore this activity is a part of all phases of product development. A key element of this is the ability to capture the engineering decisions when they are made, by storing each version of the requirements.

Product configurations are the second part of a full product representation. Configurations are tree structures that contain either conceptual sketches of subsystems and components or full CAD definition of these subsystems and components. A product definition editor tool for accomplishing the definition and conversion of system requirements into subsystems and their requirements as well as the capture of knowledge and decisions is in storyboard form in Fig. 1.

The left hand workspace in the tool provides a hierarchical textual tree where the system requirements can be specified as well as the subsystems and their requirements can be defined. In this case, a micro air vehicle (MAV) is being designed and the system requirements have been defined. The first subsystem has been identified as the lift subsystem.

Directly below the requirements tool is a library tool that provides search capability for existing sub-systems that can be used directly or used as references for the development of the new product's sub-systems. In this case, the user has searched on "lift systems" and has discovered elevator systems and flight systems. Other existing subsystems are also available for search such as sensor and payload systems.



PRODUCT DEFINITION EDITOR

INTEGRATE REQUIREMENTS, CONFIGURATIONS, AND EXPERIENCE DOMAINS

REQUIREMENTS

SYSTEM
SUB-SYSTEM
COMPONENT

MAV SYSTEM
PAYLOAD 20LBS
DURATION 2 HOURS
SENSOR VIDEO 2OZ
RANGE 1/2 MILE
LIFT SUB-SYSTEM
PROPELLOR COMPONENT

LIBRARIES

LIFT SYSTEMS
ELEVATOR SYSTEMS
FLIGHT SYSTEMS
SENSOR SYSTEMS
VIDEO SYSTEMS
INFRARED SYSTEMS
PAYLOAD SYSTEMS
ELEVATOR SYSTEMS
FLIGHT SYSTEMS

MAV SYSTEMS

ELEVATOR SYSTEMS

EXPERIENCE DOMAINS

CONFIGURATIONS

MAV SYSTEM
LIFT SUB-SYSTEM
PROPELLOR COMPONENT
SHROUD COMPONENT

LIBRARIES

LIFT SYSTEMS
ELEVATOR SYSTEMS

Fig. 1: A storyboard of a product definition editor tool.

3.2 Experience Domains

Company experience needs to be characterized and documented in a form that can be reused. These domains include successful and unsuccessful subsystems previously developed as well as technologies and materials used in the past. In order to capture this kind of information, plots are selected which best characterize the key decisions made in the design of these previous systems.

The workplace in the middle of the screen provides room for reviewing experience domain information. Experience related to specific designs is best captured as experience domains. These are typically 2D or 3D plots of trade-offs that are important to the system or components. In this case, the user has decided to explore the experience domain of elevator lift systems and a 2D plot of speed versus load appears. Also regions in the plot are identified with respect to the different types of elevator solutions used to address the need to lift loads at certain speeds.

The upper plot is a 2D plot of specific fuel consumption and load/hour versus year of development. It is an experience domain for the entire MAV system. The system design tool requires that the new system be identified on this experience domain plot. Most experience domains are made up of several tradeoff plots.

Once the previous subsystems and their associated experience domains have been reviewed, if a suitable one is found, it can be "cut" and "pasted" into the upper hierarchical text based representation of the system and subsystem requirements. If no suitable existing subsystem is found, then it must be defined from scratch.

3.3 Product Configuration

Ultimately, the purpose of this tool is to define system requirements and a system configuration that meets those requirements. The right hand column in the system design tool therefore provides a space for constructing product configurations. There are several levels to the fidelity of this configuration information. At the lowest fidelity level, there are simple sketches of the components and subsystems. At the highest level, full product documentation including CAD models and drawings are available. A typical product configuration will contain components and subsystems at various levels of fidelity. Some will be fully defined and others only concepts. In this case, the MAV system has been decomposed into a series of subsystems and component concepts.

3.4 Libraries

Each side of the tool provides library capabilities that give access to former requirements and configurations. This allows the design team to explore existing products for imitation or incorporation into the new product definition. These databases can also be refined to identify "preferred" requirements and configurations.

4. PROCESS DEFINITION EDITOR

This element provides a framework to track and aid the process of converting system requirements and configuration into an executable product development process that will produce the product deliverables.

4.1 Process Representation

Process representations are composed of two parts: process requirements and process configurations or workflows. The framework for documenting and representing processes involved in designing and producing a product is shown in Fig. 2.

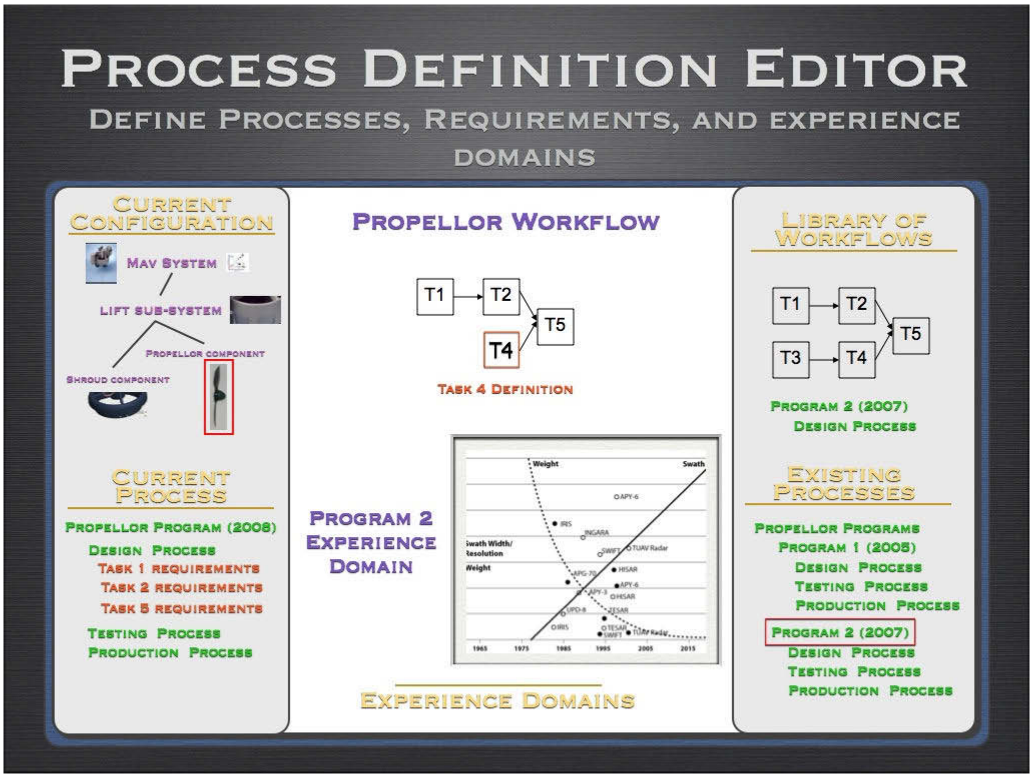


Fig 2: A storyboard of a process definition editor tool.

The left-hand side of the tool shows the product configuration that was defined earlier. Below it is a section where the current process requirements can be defined. On the right-hand side there are areas for searching existing workflows and process requirements. The middle section of the tool provides area for reviewing experience domains associated with existing processes. The upper portion of the middle section provides a workflow editor that allows tasks and sequencing to be defined for the given system, subsystem, or component being examined.

4.1.1 Process Experience Domains

Process information can be captured in regular experience domain plots where trade-offs between process capability and cost, time or other criteria. This provides a method for processes to be selected and aggregated into a specification of all the tasks needed to create the selected product.

4.1.2 Process Libraries

These libraries, much like the product definition libraries, contain former processes in terms of the stated requirements, experience domains and workflows.

5. AGENT EDITOR

The third Framework deals with the management and definition of agents. This tool allows in-house and subcontractor agents to be defined that provide the capabilities to execute the processes defined in the former tool. The agents are defined by assigning actors to specific tasks in the product development process. Usually actors are able to execute many of the tasks and so an agent is often assigned all the different tasks that the actors can do. The purpose of this is to create a library of agents that can be mapped into the different deployments representing the specific process of interest.

5.1 Agents

Most companies have extensive lists of contractors and subcontractors that they have used to complete product development tasks in the past. Often there are preferred lists made up of subcontractors who the company has chosen as the contractors they most like to work with. For the purposes of this knowledge framework, each of these subcontractors, along with the in-house employees, is considered to be actors. Actors can be a single individual, a group or company, or an automated module. Since the specific combination of actors used in a product development process represents a deployment and also dictates the success of the process, it is important to catalog each possible actor and the tasks that they can execute. Also metrics must be defined and tracked for these agents so that prior performance can be used to determine which actors will be best for a new development program.

It is important to note that actors become agents when they are assigned to a specific task. Since actors can do many of the tasks in a development program each actor will be assigned multiple tasks or all the possible tasks that they can do. The next tool or framework provides mapping capabilities that will explore all the possible combinations of agents for executing a given development program.

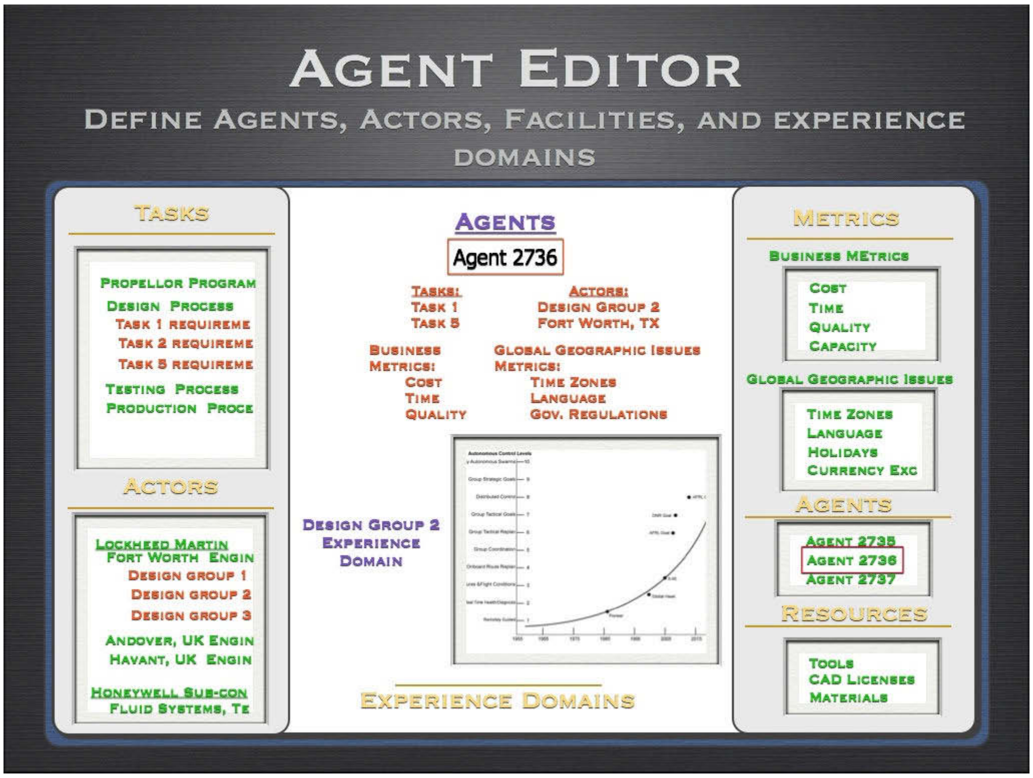
Fig. 3 shows the storyboard of a tool for defining and managing agents. Tasks and actors are selected on the left side to define an agent. Metrics on the right side can then be assigned for tracking purposes. A library of former agents is available for reference and reuse as well.

5.2 Experience Domains

Experience domains for agents consist of prior performance in terms of cost, quality, time, etc. Plots of these metrics with associated data of where each agent has performed in the past provide information that can be used to determine which actor to assign to which tasks. The information contained in the experience domains becomes a profile of each actor in terms of what they can do, what they do well, and at what level of quality and cost they usually perform.

5.3 Libraries

Once agents have been defined for the specific product development program, these agents are placed into a library that will serve as the resource for determining possible deployment options. Each development program will have its own library of agents since the choice of agents is determined for a specific development program. An actor assigned to a specific task and functioning as an agent in one development program may not be preferable in a different development program.



AGENT EDITOR

DEFINE AGENTS, ACTORS, FACILITIES, AND EXPERIENCE DOMAINS

TASKS

PROPELLOR PROGRAM
DESIGN PROCESS
TASK 1 REQUIREME
TASK 2 REQUIREME
TASK 5 REQUIREME
TESTING PROCESS
PRODUCTION PROCE

ACTORS

LOCKHEED MARTIN
FORT WORTH ENGIN
DESIGN GROUP 1
DESIGN GROUP 2
DESIGN GROUP 3
ANDOVER, UK ENGIN
HAVANT, UK ENGIN
HONEYWELL SUB-CON
FLUID SYSTEMS, TE

AGENTS

Agent 2736

TASKS:
TASK 1
TASK 5

ACTORS:
DESIGN GROUP 2
FORT WORTH, TX

BUSINESS METRICS:
COST
TIME
QUALITY

GLOBAL GEOGRAPHIC ISSUES METRICS:
TIME ZONES
LANGUAGE
GOV. REGULATIONS

DESIGN GROUP 2 EXPERIENCE DOMAIN

EXPERIENCE DOMAINS

METRICS

BUSINESS METRICS
COST
TIME
QUALITY
CAPACITY

GLOBAL GEOGRAPHIC ISSUES
TIME ZONES
LANGUAGE
HOLIDAYS
CURRENCY EXC

AGENTS
AGENT 2735
AGENT 2736
AGENT 2737

RESOURCES
TOOLS
CAD LICENSES
MATERIALS

Fig. 3: Storyboard for defining and managing agents.

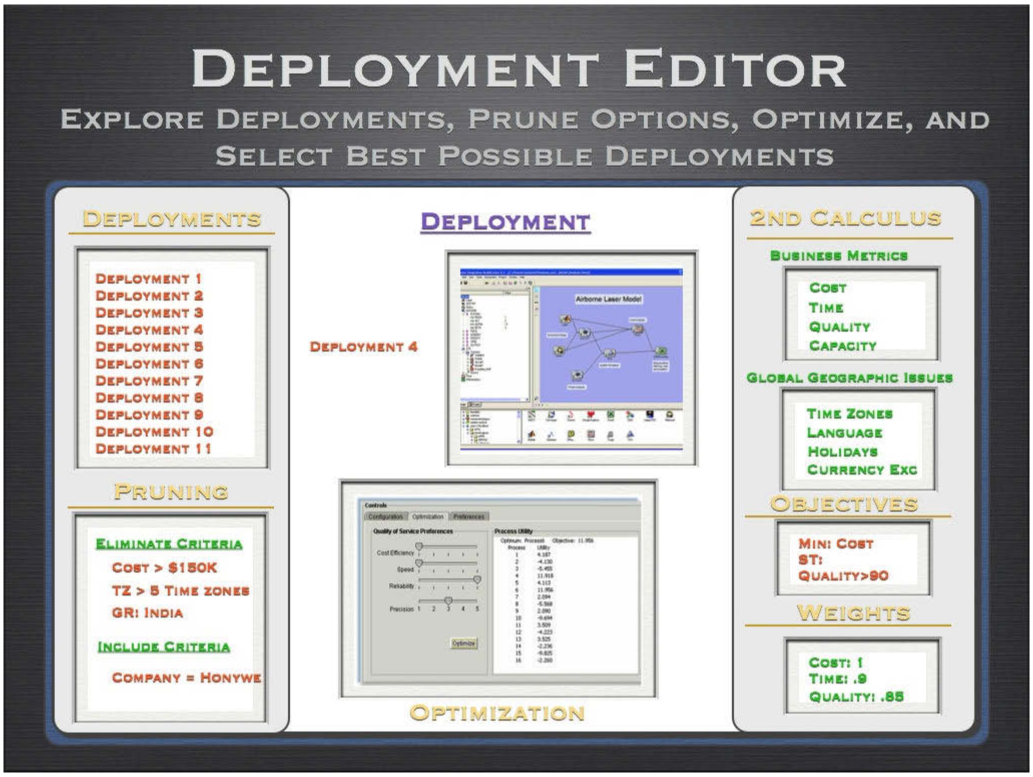
6. DEPLOYMENT EDITOR

The final framework or tool is the deployment editor. This tool is used to explore all the possible combinations of agents that can fully execute the product development process. This involves backwards mapping or deterministic mapping of all possible agents that can execute the tasks in the development program. Since there are many different possibilities, there will be many deployments. Each deployment will have unique benefits and drawbacks. A secondary calculus is therefore defined across the graph of the deployments to score each one with respect to business goals of time, cost and quality.

6.1 Deterministic Mappings and Deployments

Once all of the former tools have been completed, a knowledge engine associated with mapping is invoked that determines all of the possible viable deployments using the library of agents created in the former step. Fig. 4 shows the deployment tool and on the left hand side of the tool there is a list of the deployments generated automatically by the knowledge engine. Often times this automatically generated list can include combinations of agents that do not necessarily make sense. Therefore below the list there is a pruning tool that allows criteria to be specified that prunes the deployment list of unreasonable possibilities.

The tool also allows the inspection of each of these deployments in the upper center section where the deployment workflows with the assigned agents can be inspected.



DEPLOYMENT EDITOR

EXPLORE DEPLOYMENTS, PRUNE OPTIONS, OPTIMIZE, AND SELECT BEST POSSIBLE DEPLOYMENTS

DEPLOYMENTS

- DEPLOYMENT 1
- DEPLOYMENT 2
- DEPLOYMENT 3
- DEPLOYMENT 4
- DEPLOYMENT 5
- DEPLOYMENT 6
- DEPLOYMENT 7
- DEPLOYMENT 8
- DEPLOYMENT 9
- DEPLOYMENT 10
- DEPLOYMENT 11

PRUNING

ELIMINATE CRITERIA

- COST > $150K
- TZ > 5 TIME ZONES
- GR: INDIA

INCLUDE CRITERIA

- COMPANY = HONYWE

DEPLOYMENT

DEPLOYMENT 4

OPTIMIZATION

2ND CALCULUS

BUSINESS METRICS

- COST
- TIME
- QUALITY
- CAPACITY

GLOBAL GEOGRAPHIC ISSUES

- TIME ZONES
- LANGUAGE
- HOLIDAYS
- CURRENCY EXC

OBJECTIVES

MIN: COST
ST:
QUALITY>90

WEIGHTS

COST: 1
TIME: .9
QUALITY: .85

Fig. 4: Storyboard of the deployment editor.

6.2 Optimization
The right hand side of the framework provides optimization and secondary calculus tools that allow scores for each of the deployments to be calculated. The purpose of these secondary calculi is to provide methods for determining the impact of issues such as language differences, time zone differences, government regulations and other global geographic issues.

An optimization engine is also invoked to provide automated selection and searching for optimal deployments. This allows various criteria to be applied so that best solutions can be determined for different scenarios. The middle lower section of the tool provides feedback about the optimal deployments for a given set of criteria.

6.3 Teams
Another aspect of the deployments is that each deployment represents a different development team and so aspects of these teams are also important to examine and explore. Human resources and organization information will also need to be connected so that team dynamics can be examined.

7. KNOWLEDGE ENGINES
Underneath the four frameworks and tools are four knowledge engines. These knowledge engines include databases and logic engines that allow for the capture of knowledge and the reasoning and search capabilities needed. These four engines must be built and implemented first and then the frameworks and tools can be developed on top of them.

7.1 Requirements Knowledge Engine
This is the most basic of the four knowledge engines. It is fundamentally a database that allows for items to be identified and then relationships to be established hierarchically as well as indirectly. This database must be able to store product requirements with all of the subsystem and component

requirements and establish the connections between them. It must also provide capabilities to store process requirements in much the same way. All of the actor and agent information will be stored in this same database structure.

Search capabilities must also be provided so that previous entries are available for reference. The search capabilities should also provide navigation of the interconnections and relationships between requirements for products and processes. It should allow search capabilities for agents and then navigation to explore the process used in one of the deployments that an agent worked on as well as the product requirements.

All of these capabilities should be achievable using a standard database and an ontological engine that sits on top of it and provides reasoning for relationships such as OWL. The items could be constructed as "tuples" so that relationships can be established and explored.

7.2 Configuration Knowledge Engine

The configuration engine is similar to the requirements engine and could possibly be incorporated into the requirements engine. The fundamental difference is that it must support a wider range of data formats since the configuration information can be expressed as simple sketches or as detailed as full CAD/CAE/CAM files providing detailed product documentation. All the same relationships will be needed as the requirements engine. The ability to invoke the native authoring tools to display and examine the configurations will also be needed.

7.3 Experience Domain Engine

The experience domain engine stores all the historical data for product, process, and agent experience. The logic part of the engine provides graphing capability to display not only the design spaces but also the historical products, process and agent that have been used. These data points must also be navigable so that hovering over the points and clicking on the points provides links to the more details experience information such as cost, time to market, field data, etc. The plots should also provide regional information to indicate regions of the design space where certain technologies are preferred. The plotting should also provide 2-D or 3-D plots and the ability to combine different data into tradeoff plots.

7.4 Mapping and Optimization Engine

This engine is the most complex of all of the engines. It must provide autonomous agent capability so that the agents identified to be used in a particular deployment can actually function as pseudo-autonomous agents in the mapping process. There are two logic capabilities related to this engine. The first is a deterministic or backwards mapping capability that involves setting up an autonomous agent environment where all the agents are able to "engage" in a pseudo-market environment. Each agent "knows" the inputs required to do its tasks and what the deliverables are. A "mapper" agent then connects the outputs and inputs to satisfy all dependencies associated with delivering the final product. The result is a graph of all the agents required to complete the given product development process. Since there are redundant agents, there will be multiple deployments of the process.

The second logic capability is used to apply secondary calculi across these deployment graphs and score the deployments as well as use these scores in an optimization to select the best possible deployments based on the given criteria.

8. CONCLUSIONS

Fig. 5 demonstrates the interaction between the editors and between the editors and the databases. Product development starts with the product definition editor. The product definition editor has access to the requirements, configuration and experience domain databases. It uses these databases to fully define the overall product and all of its subsystems. With the product fully defined, the definition is passed to the process definition editor. This editor accesses the requirements and experience domain databases to define the process for producing the product. The process is then passed to the agent editor, who accesses the experience domain database and agent library to define who can complete the processes. This information is then passes to the deployment editor that accesses the agent library and deployment database to define different deployment for the process.

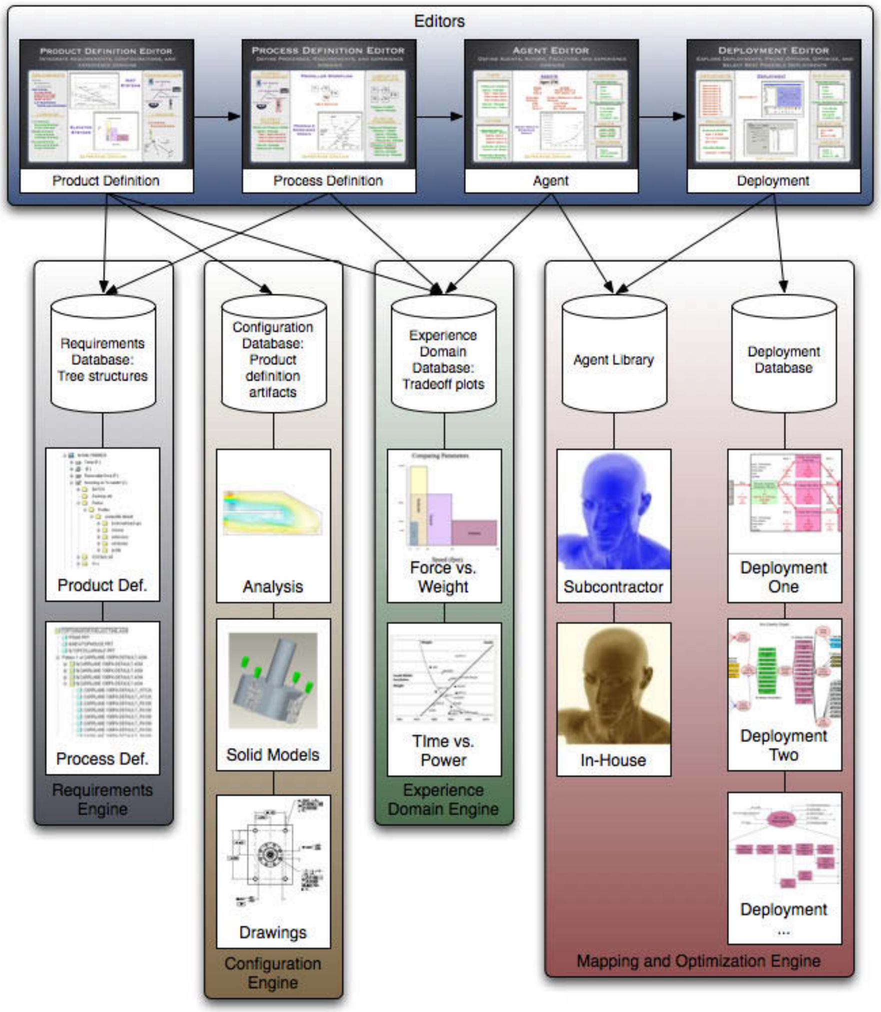


Fig. 5: Process overview.

This process resolves the above-mentioned problems. First, the problem that knowledge capture is time consuming and not incorporated into the engineering process is resolved by the automatic capture of knowledge imbedded in the system and the use of the system is to design the product. The second problem, usability, is resolved because all knowledge is captured in the context in which it was created. This is demonstrated in part because the process may be executed, thus all necessary knowledge must be captured. The final problem of passing knowledge between generations of employees is also resolved by the capture of knowledge in the context in which it was created.

The implementation of this strategy involves the creation of the four knowledge engines, the populating of the data sets with historical information from the company, and the creation of the framework tools that facilitate the design of executable plans for product development. After the successful implementation of the knowledge engines and the tool deployment, significant work will be needed to change the corporate culture of preliminary design and product development. These tools will provide more information and search capability than has ever been used for planning a product development deployment. Many of the difficult aspects will be associated with proscribing a new human process and implementing it through training and management.

9. REFERENCES

[1] Ajit, S.; Sleeman, D.; Fowler, D. W.; Knott, D.: Constraint capture and maintenance in engineering design, Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 22, 2008, 325-343.

[2] Baxter, D.; Gao, J.; Case, K.; Harding, J.; Young, B.; Cochrane, S.; Dani, S.: An engineering design knowledge reuse methodology using process modeling, Res Eng Design, 18, 2007, 37-48.

[3] Boeri, R. J.: Searching for KM, EContentMag, January 2003, http://www.econtentmag.com

[4] Bohm, M. R.; Stone, R. B.; Simpson, T. W.; Steva, E. D.: Introduction of a data schema to support a design repository, Computer-Aided Design, 40(7), 2008, 801-811.

[5] Fruchter, R.; Demian, P.: CoMem: Designing an interaction experience for reuse of rich contextual knowledge from a corporate memory, Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 16, 2002, 127-147.

[6] Gotel, O. C. Z.; Finkelstein, C. W.: An analysis of the requirements traceability problem, Requirements Engineering, 1994., Proc. of the First International Conference on, 1994, 94-101.

[7] Hwang, W.; Salvendy, G.: Effects of an ontology display with history representation on organizational memory information systems, Ergonomics, 48(7), 2005, 838–858.

[8] Kalkan, V. D.: Knowledge Continuity Management Process in Organizations, Journal of Business and Economics Research, 4(3), 2006, 41-46.

[9] Kitamura, Y.; Washio, N.; Ookubo, M.; Koji, Y.; Sasajima, M.; Takafuji, S.; Mizoguchi, R.: Towards a Reference Ontology of Functionality for Interoperable Annotation for Engineering Documents, Proc. of Posters and Demos of the 3rd European Semantic Web Conference, 2006, 75-76.

[10] Komi-Sirviö, S.; Mäntyniemi, A.; Seppänen, V.: Toward a Practical Solution for Capturing Knowledge for Software Projects, IEEE Software, 2002, 60-62.

[11] Li, Z.; Ramani, K.: Ontology-based design information extraction and retrieval, Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 21, 2007, 137-154.

[12] Nanda, J.; Simpson, T. W.; Kumara, S. R. T.; Shooter, S. B.: A Methodology for Product Family Ontology Development Using Formal Concept Analysis and Web Ontology Language, Journal of Computing and Information Science in Engineering, 6, 2006, 103-113.

[13] Sim, S. K.; Duffy, A. H. B.: Towards an ontology of generic engineering design activities, Res Eng Design, 14, 2003, 200-223.

[14] Udeaja, C. E.; Kamara, J. M.; Carrillo, P. M.; Anumba, C. J.; Bouchlaghem, N. D.; Tan, H. C.: Live Capture and Reuse of Construction Project Knowledge: Capri.net Approach, Automation in Construction, The Joint International Conference on Construction, Culture, Innovation and Management (CCIM), The British University in Dubai, UAE, 2006, 813-823.