A METHODOLOGY FOR KNOWLEDGE DISCOVERY AND CLASSIFICATION

Janis Terpenny[1], Stephen Strong[2], Jiachuan Wang[1]

[1]Department of Mechanical and Industrial Engineering
University of Massachusetts
Amherst, MA 01003-2210

[2]Apprentice Systems, Inc.
15021 21 Drive, SE
Mill Creek WA, 98012

ABSTRACT: Knowledge-based engineering (KBE) has emerged as a valuable tool for many industries seeking to harness company information and knowledge resources. Commercially available KBE systems offer a variety of development languages and interfaces for this purpose. While offering the potential for great savings, these systems frequently require substantial setup to be successful. Companies that do not succeed with KBE technologies fail either because of excessive time invested with limited return or due to lacking knowledge engineering skills. Applying ones knowledge comes as second nature to many. Discovery, describing, and capturing this knowledge is difficult. This paper presents a methodology that has been developed to assist the discovery, classification, and capture of design knowledge. An example problem is included and demonstrates the eleven steps of the methodology. It is suggested that this methodology will assist the knowledge mining requirements for implementation of applications in any KBE system.

INTRODUCTION

Background and Motivation

Few would argue that information and knowledge are crucial to engineering design. Designing products, systems, and/or processes can require information and knowledge from a great number of resources (internal, external, digital, and otherwise). Of late, knowledge-based engineering (KBE) has emerged as a valuable tool for many industries seeking to harness and perhaps integrate company information and knowledge resources so as to automate and/or assist decision-making processes. Knowledge reuse, standardization, reduced time-to-market, and improved quality are often cited as the benefits of such systems.

Knowledge collection supporting engineering design tasks at the early stage of the design process is difficult, especially when the knowledge is coming from a great number of resources. In industry, from function modeling, entity-relation information modeling to object-oriented modeling, individuals and companies are more and more concerned about how to capture knowledge and manage it efficiently.

Intuitively, some knowledge capture methods have been defined using structured natural language, such is the case for G2; developed by the Gensym Company[1]. G2 is a commercial product for building applications for business operations. It uses structured natural language to capture knowledge with rules and procedures. Although it claims to enable even non-programmers to read, understand, and modify applications, it is not a formal way to systematically organize and classify data and knowledge.

A more formal method to facilitate knowledge collection is UML [2] (Unified Modeling Language). UML is a graphical language for analyzing, specifying, designing, constructing, visualizing, and documenting the artifacts of a software-intensive system. But with UML, non-technical end-users, managers and business domain experts find it difficult to understand UML visual models. This may lead to problems in knowledge capture from user and domain experts.

An important insight obtained from system modeling is that any modeling process, by definition, implies a modeling system one level higher. Such meta-modeling systems [3-4] represent and specify the requirements to be met by the modeling process. Meta data, which stands for data for data, is about basic concepts and terms especially for a specific domain. Meta modeling is also referred to as ontology or repository construction. For example, IDEF5 [5] is such an ontology description capture method. It is concerned with building a catalog of terms used in a domain. Ontology is similar to a data-dictionary but includes both a grammar and a model of the behavior of the domain. Nevertheless, the graphical representation of ontology mentioned in IDEF5 is too general, not very applicable to building real systems. Microsoft Repository [6] is an example of an object-oriented meta-data management facility. It tends to be somewhat complex since it is intended to accommodate users from a wide variety of areas.

The Problem

Commercially available KBE systems are on the rise offering a variety of development languages and interfaces to capture and make use of company knowledge. While offering the potential for great savings, these systems frequently have required large investments in time from information management personnel and/or automation experts, and senior designers within the company who possess high-level expertise and knowledge. Predominately, the companies that fail at implementing and succeeding with KBE technologies are those who do not realize the significant investment of time and knowledge engineering skills that are required to succeed. Applying ones knowledge comes as second nature to many. Discovery, describing, and capturing this knowledge is difficult. To be successful, and to reduce the time investment in setting up and implementing KBE applications, users must be guided and supported in the processes of knowledge discovery and knowledge capture.

Overview

This paper presents a methodology that has been developed to assist the discovery and capture of design knowledge. The Knowledge Capture Methodology (KCM) outlined in this paper provides clear steps to examine successful products, decompose knowledge, capture and classify needed information. An example problem is included and demonstrates the steps and utility of the methodology. While this paper discusses the methodology within the context of a particular KBE engine, Apprentice System [7], KCM will assist the knowledge mining requirements for implementation of applications in any KBE system.

APPROACH

The terminology and concepts presented in KCM have evolved over the last decade based on discussions with practicing designers in many disciplines. The methodology is independent of any commercial system implementation and has proven to be invaluable during the design and specification stage of engineering automation tools. It is an archeological approach to knowledge discovery and capture. It focuses on examining real designs and identifying the features and traits that make them successful. The methodology organizes this knowledge into reusable software components that can be assembled automatically by an Apprentice™ (or any software agent that makes design decisions based on learned rules). As proven in application, capturing the knowledge of just a few designs allows a KBE engine such as Apprentice™ to generate designs correctly simply by adjusting parametric geometry, applying constraints, and combining options.

KCM is composed of 10 steps that are repeated in a cycle (Step 11). Each journey around the cycle produces new knowledge. Briefly, Step 1 identifies the domain. Steps 2 and 3 focus on the discovery of product data attributes. Steps 4 and 5 leverage object oriented techniques to classify components. Steps 6, 7 and 8 identify subjective knowledge about the use or application of each component. Steps 9 and 10 integrate the data, object, and models into a knowledge-base tied together through visualization. A more detailed description of the 11-step methodology is presented next and then followed by a detailed example problem.

Knowledge Capture Methodology

The knowledge capture methodology described here provides a procedural process to organize thoughts, promote communications, and uncover 'why we do the things we do'.

Step 1: *Select a product or process to model.* This sounds simple but takes some discipline. Select a physical product instance to reverse engineer and build the knowledge model. This will provide something to refer to when answers to tough questions are needed. It also provides a focus on a demonstrable deliverable. Consider 4 other variations of this product that will bring breadth to the knowledge model in later iterations.

Step 2: *Decompose the product into atomic components.* Atoms are the lowest level components that are used to define a product or process. Use a bill of materials to help identify which components are atomic. If a subassembly is purchased from another supplier, consider it atomic and self-contained. These atoms will be the building blocks of all future product definitions. At this early stage it is only important to identify object instances and name them, consolidating components into groups (classes) based on similar looks would be premature.

Step 3: *Assign attributes to atomic components*. Each atom has a set of characteristics that make it unique from other atoms. Identifying which attributes are related to an atom comes from asking questions related to what sort of information this atom should communicate within a model. What is your part number? What is your volume and weight? Many different atoms have common attributes. Try to reuse attribute names often. If the goal is to build a product configuration knowledge base, identify characteristics that pertain to component selection.

Step 4: *Establish atomic instances in database or tabular data sour*ce. Sort each atomic instance into a tabular data source or database. Use the property definitions to define the field names and populate the table with the actual data values from the product under scrutiny. Each record should be unique and have a key field or part number that identifies that data record uniquely.

Step 5: *Create component classifications for the atomic instances*. Component classification allows sharing of attributes and validation rules. Classification makes it possible to execute logical inferences and configuration decisions based on component types. Review the atomic components and use common attribute names to help identify a classification scheme. The tendency is to classify atoms based on themes and group items that have the same shape or behavior. The best classification schemes evolve over time and tend to be organized around component utility or application in the more abstract classes and physical attributes in the more specific (or lower) classes. One final note: it is best if all classes originate from a central root class. This makes it possible to influence all the components simultaneously.

Step 6: *Create use cases by grouping atomic components into assemblies*. The process of building assemblies from classified components is how to collect use-case knowledge. It is through the application of a component that the functional characteristics, or uses, become obvious. When an assembly is built, a product model is defined that describes which components are subparts of other components. The "has parts" / "subpart of" relationship between components is the fundamental organizational relationship for the semantic model that describes an assembly tree. During this step it is important to name each component in the assembly, but it is not necessary to name every component in the assembly uniquely, only those that share the same "subpart of" or parent component. Having unique names within an assembly allows the agents to refer to each component without confusion. In naming the overall assembly, or use case, consider why it was constructed. What function does it serve? What is the function of sub-components and how do they contribute to the performance goals of the assembly? Also consider, are there alternative components to the ones selected that are functionally equivalent?

Step 7: *Introduce a use case for each existing component classification*. Every component has an intended use and many components have multiple uses. A component's application is easy to see when it is part of an assembly. Selecting the correct component type for an application is a problem solving discipline. Capturing this knowledge of potential component uses will allow an agent to make product configuration decisions automatically. Linking a component definition with the knowledge of how it can fulfill a design requirement is the single most valuable piece of knowledge that can be collected. It addresses under what particular conditions, or situation, the component (whether an assembly or low-level component) can be used.

Step 8: *Define a set of relations and create relationships between considerations*. The default relationships of "has parts" / "subpart of" were used to organize the product model into an assembly. This organizes the product into a tree structure of indented bill of materials. Most sub-assemblies have their own internal organization that can only be described as a network created by a set of custom defined relationships that link the sub-components together. These relations can define spatial constraints, chronological constraints, containment, dependencies, or anything else the user defines. Many relationships have inverses that are implied implicitly, this is also an important part of the knowledge capture process.

Step 9: *Use relationships between components to propagate parametric values and apply selection constraints.* Relationships between components make is possible to extract parametric values from an associated component without knowing its name. By default, the communication between the components (the direction of parameter propagation) runs up and down the tree from parent component to sub-component along the relationships. A custom relationship name can be used to redirect the value query to another component.

Step 10: *Associate parametric values of components for reporting and visualization.* Every component has at least one graphic presentation. Presentations are part of the component's definition

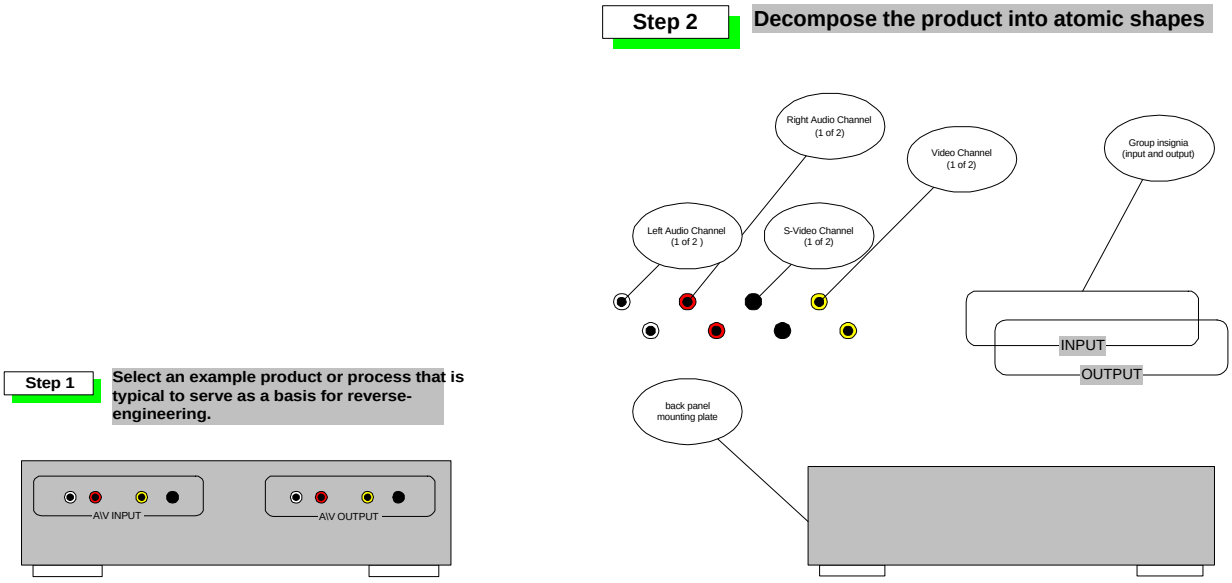
and therefore use rules to tie their parameters to the component being presented. All atoms, relationships, assemblies, and definitions documented in the previous steps are now in a generated model.

Step 11: *Repeat the process*. Further refinement of atoms, assemblies, attributes, situations and relationships are needed to build a rich KBE environment. This will come with additional iterations and new product considerations.

EXAMPLE PROBLEM USING KCM

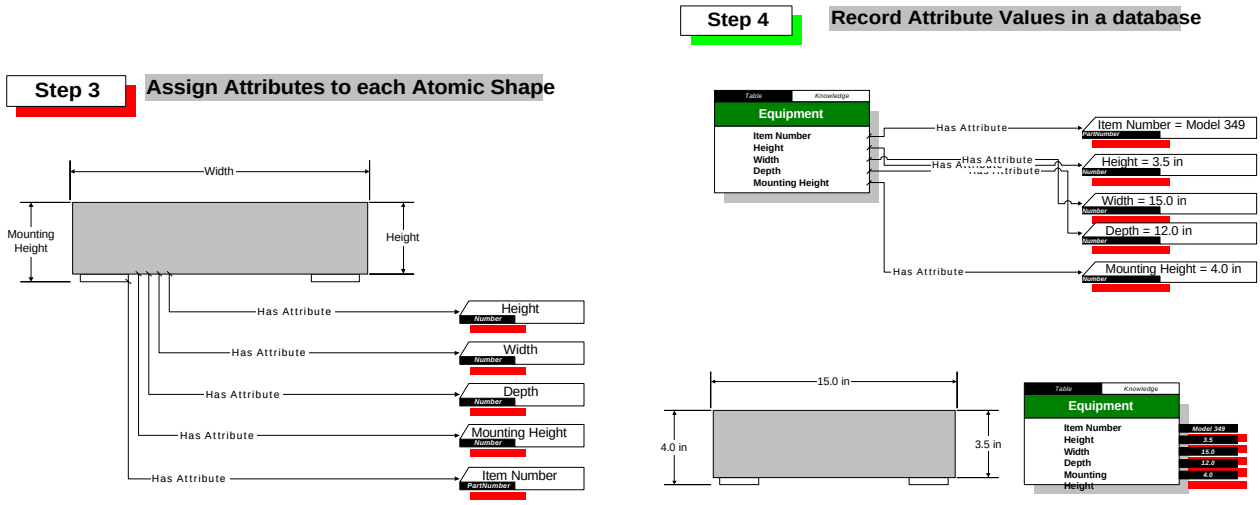
Consider the specification, design and configuration of an electronic home entertainment system. This industry is full of technical specifications that dictate component compatibility and design decisions. A typical system consists of 12 to 15 components, not including specialized cabling. These components "fit" together based on a set of evolving standards. For the purpose of our example, we will concentrate on automating the process of schematic generation and cable selection. Applying the KCM will reduce the complexity of thousands of designs into a few simple diagrams. Furthermore, we will build a foundation for the Apprentice™ to auto wire systems of components and even learn from practitioners. Steps 1-10 of KCM are shown in a series of diagrams below.

Step 1 shows the selection of a product. The back panel of a VCR is our starting place. This is a simple, yet rich example. Step 2 involves decomposing the VCR back panel into a mounting plate, two group insignias and pairs of left audio channels, right audio channels, video channels and s-video channels. From observation, these are the building blocks of a VCR back panel. It does not make sense to decompose the audio channel into to circles and label, or the group insignias into four lines, four rounded corners and a label.

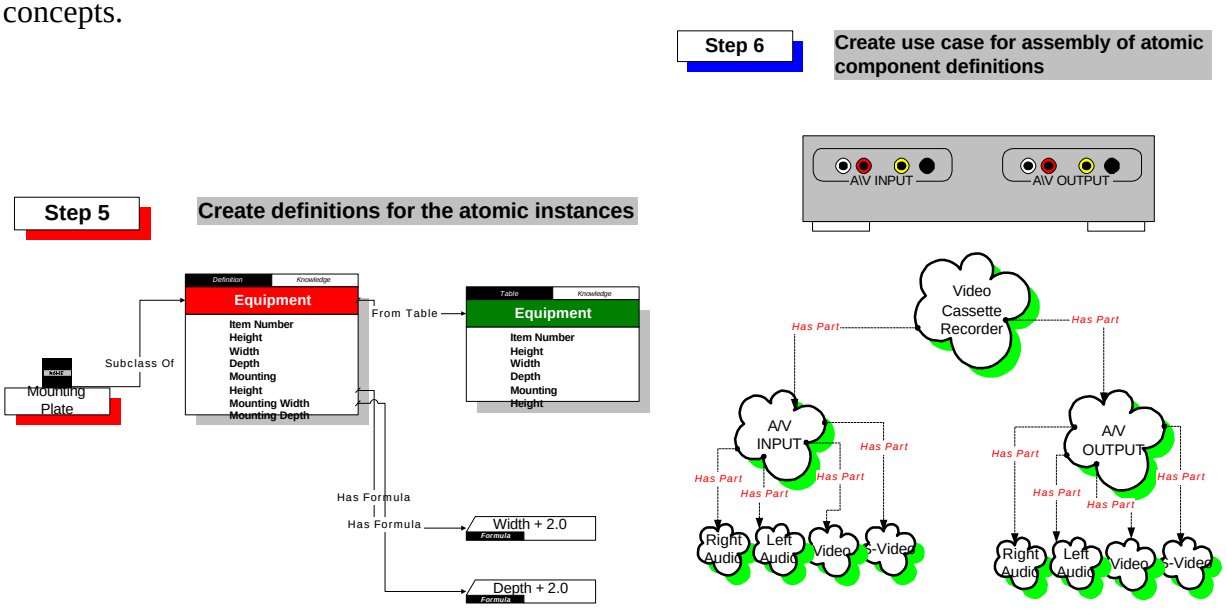


For Step 3, assign attributes to atomic components, a number of properties have been defined, Item Number, Height, Width, Depth, Mounting Height, Label, Connection Type, Connection Signal, X Location, and Y Location. Each property is assigned a data type and many of the properties are associated with multiple atoms. Reusing property names makes it easy to maintain a product data dictionary. In most applications, these attributes become custom properties or user cells. They are used to collect user specifications and drive shape behavior. It is not necessary to make the decision how to implement the attributes at this stage, only to identify them. In fact, rushing to define custom

properties and user cells usually means rebuilding the shapes later. In Step 4, establish atomic instances in database or tabular data source, the actual data values were taken from the mounting plate and loaded into a database. They are stored in a table named “Equipment” and identified with the key field “Item Number”. Not every atomic instance needs to be placed in a database. The parameters of the group insignias can be entirely calculated using a formula. Step 4 is quite significant in that the attributes affecting decision-making have been identified, a schema for product data management is in-process making the tools available for part rationalization.

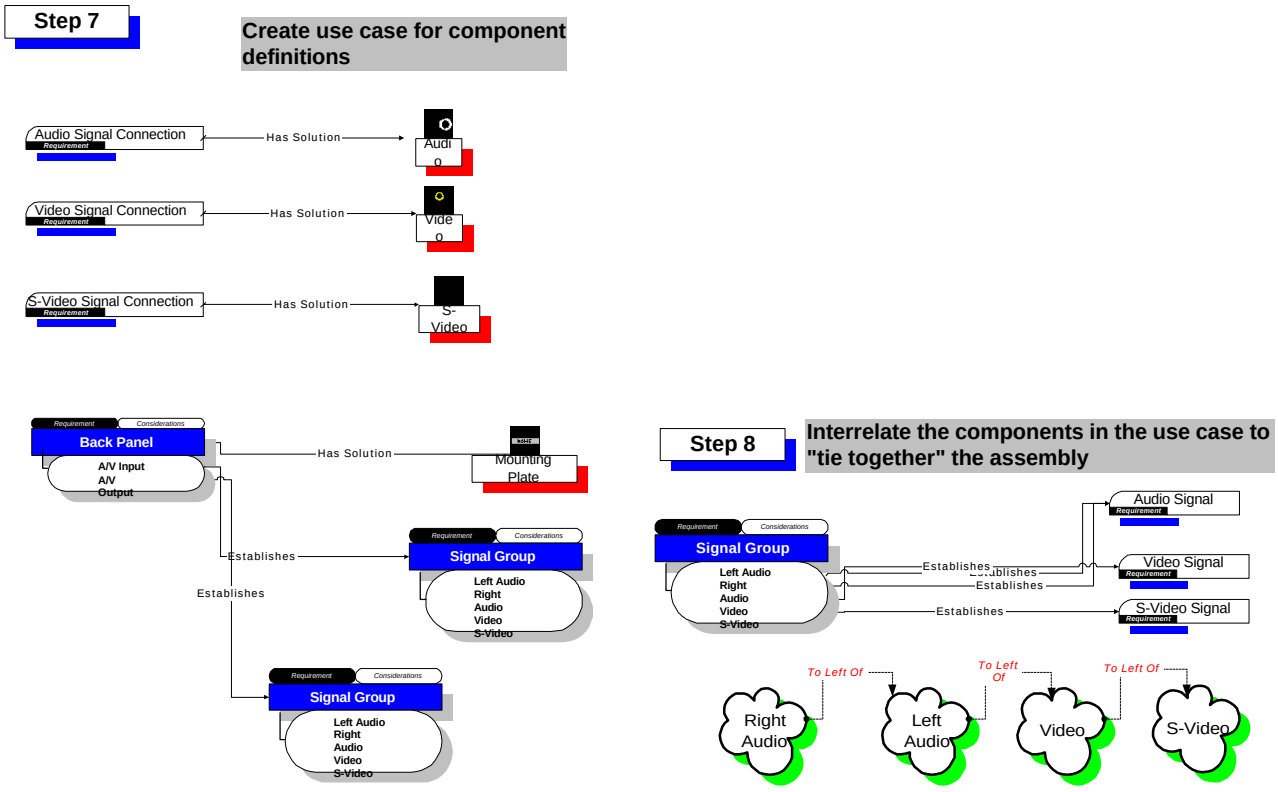


A portion of Step 5, create component classifications for the atomic instances, is shown in the diagram. Three abstract classes of components were created to organize the existing atoms: Equipment, Terminal, and Signal. The Equipment definition was established and associated with the Equipment table. A mounting plate is defined as a subclass of Equipment and therefore it inherits all the attributes of Equipment including the constraints that force the attributes to take on the values of one record in the table. Notice that the definition contains formulas for the calculation of the mounting width and mounting depth. The Terminal definition is used to group and later identify components and behave like terminals. S-Video is a special type of video terminal. The Signal Group definition is the basis for two grouping types. To reinforce each type the attribute Signal Direction has been added and is limited to the values Input and Output. The impact of Step 5 is significant in that with the component class definition provides an entire universe of parametric designs. The product data defined earlier now becomes a set of real recorded alternative designs. Adding class attributes, methods and rules makes it possible to amplify core design data and explore new design concepts.

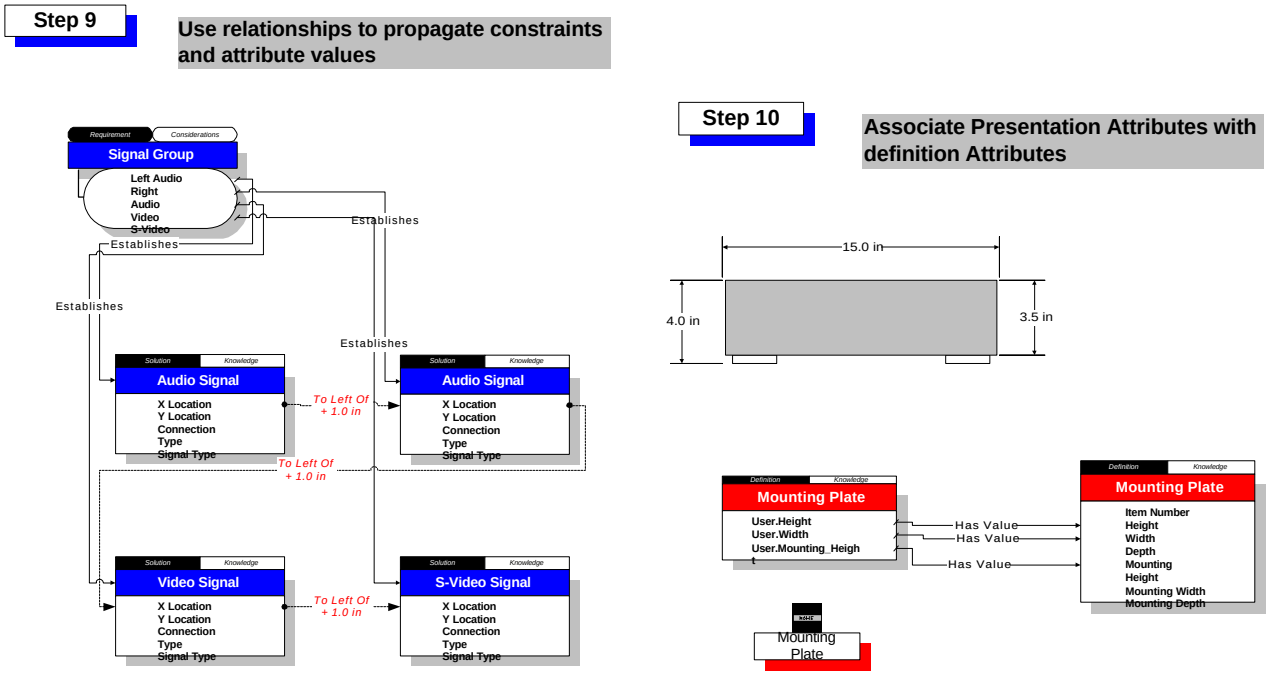


In Step 6, create use cases by grouping atomic components into assemblies, the A/V INPUT signal assembly is created by grouping together each one of the connection components and an input signal group insignia. This allows each connection component to refer to is parent component to determine its connection direction. This is important because the value of a connection component's connection direction is dependent on how it is used and can only be determined in application. An A/V OUTPUT signal assembly is identical to the A/V INPUT signal assembly except the input signal group is replaced with an output signal group. After completing Step 6, components are assembled to fulfill a design goal, or more importantly, a customer requirement. Think of product configuration management as solving a set of problems to meet a series of customer requirements. This view will insure that knowledge has been collected to generate custom designs automatically.

Step 7 introduces a use case for each existing component classification. Mapping each component definition to an intended application is how use case knowledge is captured. The terminology describes how a requirement is fulfilled by a solution (or component definition). The requirement for an Audio Signal Connection can be fulfilled by an instance of an audio component definition. The requirement to create a back panel can be fulfilled by creating an instance of a mounting plate and then establishing an A/V Input and an A/V Output both by fulfilling the Signal Group requirement. The requirement/solution knowledge allows decomposition of the product into a series of smaller design problems. The Apprentice agent manages the complexity of all the use cases and design scenarios. After completing Step 7, every component definition has a documented design purpose. There is accountability for every design decision and an audit trail for every component in the product model. In addition, definitions that share the same purpose are functionally interchangeable. For Step 8, define a set of relations and create relationships between considerations, two spatial relationships exist between the components used in the signal group requirement. The left audio, right audio, video, and s-video components are positioned side-by-side. Each component needs to know the component to the left and to the right of itself. Furthermore, each component needs to know that they are to be contained within the signal group.



Step 9, use relationships between components to propagate parametric values and apply selection constraints, the graphic representation of the formula or constraint can be quite elegant. The x location of the left audio is constrained to be the x location of the right audio + 1.0 in. The x location of the right audio is constrained to be the x location of the video + 1.0 in. The constraint is created by linking together the attributes for each component and extending the "equals" formula as necessary. In Step 10, associate parametric values of components for reporting and visualization, two different views of the mounting plate definition are shown to detail how the constraint is placed between the presentation parameters (user.height, user.width, user.mounting_height) and the selection parameters (height, width, mounting height).



SUMMARY AND CONCLUSIONS

This paper has presented a methodology for knowledge discovery and classification. Specifically, the 11-step methodology (KCM) has been described and demonstrated through a detailed example problem. The benefits of the methodology are numerous.

Components and relationships (a semantic model) is the way humans naturally organize and reason about the world around them. It is only logical, as the approach described herein, to approach the construction of models of products or processes similarly. Using KCM, every component has a documented purpose, a derived classification and a known set of relationships to other components in the model. Relationships between components constrain their parameters so only the set of feasible designs that meet customer specifications are generated. It is possible with this approach to have the bill-of-materials, component part number selections, assembly instructions and process plans all generated by querying the product model. Relationships can structure the model to account for virtually every perspective within an organization.

In summary, the KCM is capable of producing and documenting four mission critical deliverables:

1. a product data schema that supports all perspectives in an organization,
2. a classification design and implementation plan (independent of system of implementation),
3. deep design knowledge of component use cases that is largely undocumented from company expert designers and only developed through years of experience, and
4. a product knowledge base that generates an audit trail for every design decision that can be used to reduce mistakes, capture exceptions, and promote learning within an organization.

REFERENCES

1. Gensym Company, http://www.gensym.com
2. The Unified Modeling Language, UML 98: Beyond the Notation, First International Workshop, Mulhouse, France, June 1998.
3. Englebert, Vincent and Hainaut, Jean-Luc. "DB-Main: A next generation meta-case," *Information Systems*, Vol. 24, No. 2, pp. 99-112, 1999
4. Nissen, Hans W. and Jarke, Matthias. "Repository Support for Multi-perspective Requirements," *Engineering Information Systems*, Vol. 24, No. 2, pp. 131-158, 1999
5. Information Integration for Concurrent Engineering (IICE) IDEF5 Method Report, Prepared for Armstrong Laboratory AL/HRGA Wright-Patterson Air Force Base, Ohio 45433, Prepared by: Knowledge Based Systems, Inc.1408 University Drive East College Station, Texas 77840, Revision Date: September 21, 1994
6. Bernstein, Philip A., Bergstraesser, Thomas, et al, "Microsoft Repository Version 2 and The Open Information Model," *Information Systems*, Vol. 24, No. 2, pp. 71-98, 1999
7. Apprentice Systems, Inc., http://www.apprenticesystems.com