Multiagent Based Product Data Communication System for Computer Supported Collaborative Design

Bernadetta Kwintiana Ane [1], **Dieter Roller** [2], **and Ajith Abraham** [3]

[1, 2] *Institute of Computer-aided Product Development Systems, Universität Stuttgart Universitätsstr. 38, D-70569 Stuttgart, Germany.*
E-Mail: ane@informatik.uni-stuttgart.de [1], *roller@informatik.uni-stuttgart.de* [2]

[3] *Faculty of Electrical Engineering and Computer Science*
VSB - Technical University of Ostrava,
17, listopadu 15/2172, Ostrava - Poruba, Czech Republic
E-Mail: ajith.abraham@ieee.org [3]

ABSTRACT

Today, designers and engineers on collaborative design environments often work in parallel and independently using different tools distributed at separate locations. Due to unique characteristic of engineering design, interaction during product development is difficult to maintain. As the information and communication technologies advance, computer supported collaborative design (CSCD) becomes more promising. Nevertheless, a potential problem remains between the product design and manufacturing, which primarily lies on the geometric shape of products that exists inherent in mass-customization. Meanwhile, each CAD/CAM technology has ist own authoring tools, which govern the use of independent language and format for expressing various features and geometry. This condition creates incompatibility and has significant impact to the product costs. This chapter is to address the incompatibility problem by introducing the architecture of a multiagent-based product data communication system. The developed system is adaptive and has a capability for autonomous tracking of design changes. The tracking model is able to support forward and backward tracking of constraint violation during the collaborative design transactions.

BACKGROUND

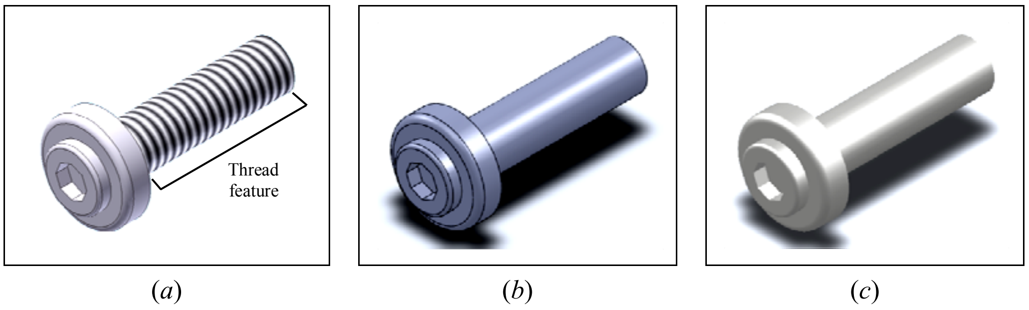
Today's industry requires massive computer-supported technologies to address the increasingly complex product development tasks and the high expectations of customers. As the information and communication technologies advance, the application of collaborative engineering to product design, so-called computer supported collaborative design (CSCD), becomes more promising.

Sprow (1992) defines CSCD, or so-called cooperative design, as the process of designing a product through collaboration among multidisciplinary product developers associated with the entire product life cycle. CSCD is carried out not only among multidisciplinary product development teams within a company, but also across the boundaries of companies and time zones, with increased numbers of customers and suppliers involved in the process.

Accomplishing a design task and delivering the results to manufacturing requires immense and complex information. Currently, most CAD/CAM technologies govern independent authoring tools in different proprietary formats. Meanwhile, a potential problem between design and manufacturing remains in the geometric shape of products, which mainly exists inherent in mass-customization. For instance, creating 'thread' on a screw using the *thread* feature operation in the Autodesk Inventor as depicted in Figure 1. This feature often cannot be recognized when the design is transferred and read using another CAD/CAM system (e.g., Solidworks). Or, in some cases it will be recognized with certain deviation of dimensions and tolerances. This condition creates incompatibility problem.

Failures in the final design requires engineer to perform design rework, which has significant impact to the product costs. Furthermore, if failures are recognized after the design being manufactured, it will result in such condition the whole products to be rejected.

Figure 1. Screw: (*a*) original design created by Autodesk Inventor, (*b*) original design read in Solidworks, (*c*) translated design to neutral file in Solidworks.



In collaborative design environment, the design – build – test cycle is performed by designers and engineers who work with various application systems in geographically distributed locations. When change is applied on a part, changing of shapes or dimensions will create constraints propagation to the adjacent parts that might affect the overall performance of the product. In this regard, the ability to tracking design changes becomes important. Therefore, the synchronization of product data communication along product development process is necessary to take place.

This chapter aims to address the incompatibility problem in the collaborative design environment. To support design exchange, a multiagent based product data communication system is introduced. The adaptive system is developed on the Cloud technology, where a shared server provides resources, software and data to designers and engineers to perform on-demand real-time collaborative work at remote locations. The data communication network is designed based on the seven-layers ISO/OSI model. The ISO/OSI network is an Open System Interconnect (OSI) model developed by the International Standards Organization (ISO) in 1984 (ISO, 1992) as a conceptual framework for communication in the network across different equipment and applications by different vendors. Today, ISO/OSI model is considered as the primary architectural model for intercomputing and internetworking communications.

In the following paragraphs, Section 2 describes the conceptual framework of computer supported collaborative design (CSCD) and the applications of Cloud Computing and agent-based technologies, Section 3 explains the architecture of the multiagent based product data communication system, Section 4 provides an illustration on the system's capability for tracking of design changes. Finally, Section 5 provides the conclusion of current work and a direction for future works.

COMPUTER SUPPORTED COLLABORATIVE DESIGN

Many researchers consider CSCD as an application of computer supported cooperative work (CSCW) in design. The term CSCW was first used by Greif and Cashman in 1984 to describe a way to support people in their work arrangements with computers (Greif, 1988; Schmidt and Bannon, 1992). Engineering design has some unique characteristics (e.g., diverse and complex forms of information, interdisciplinary collaboration, and heterogeneous software tools), which make interactions difficult to support. Therefore, design has become one of the most important applications of CSCW technologies.

According to Shen, Hao and Li (2008), an important objective of CSCD is to address the insufficient or even absent manufacturability checks concurrently by detecting and considering conflicts and constraints at earlier design stages. To support collaborative design, information and communication technologies are used to augment the capabilities of the individual specialists, and enhance the ability of collaborators to interact with each other and with computational resources.

With the rapid advancement of Web-based technology, CSCD has progressed dramatically. The depth and breadth of CSCD applications are far beyond the traditional definition of concurrent engineering. Moreover, the application of agent-based technology has been investigated to be effective to enhance communication, cooperation and coordination amongst design team as well as software tools. Today, the need to dynamically provide resources of various kinds as services which are provisioned electronically in the collaborative design environment has lead to the use of Cloud Computing. In this framework, services should be available in a reliable and scalable way so that designers and engineers can use them, either explicitly upon request or simply as and when required.

Web-based Technology

The Web was originally developed for information sharing within internationally dispersed teams and the dissemination of information by support groups. Since its emergence in the early of 1990s, the Web has been quickly adopted into the collaborative design environment to publish and share information relevant to the design process, from concept generation and prototyping to virtual manufacturing and product realization. A CSCD system developed with the Web as a backbone will primarily provide access to catalogue and design information on components and sub-assemblies, communication amongst multidisciplinary design team members, and authenticated access to design tools, services and documents (Shen et al., 2008). The Web can be integrated with other related technologies and used for product data management.

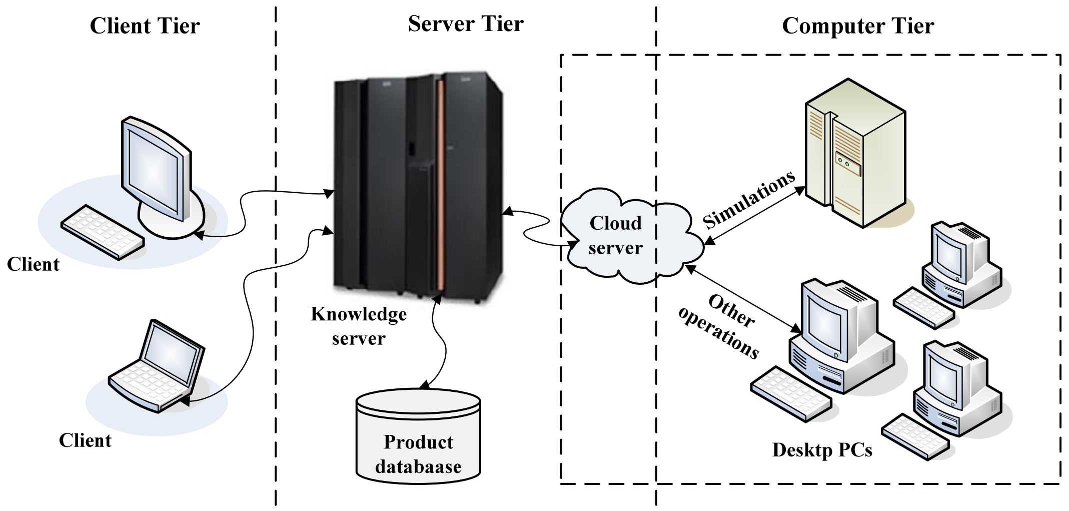
Along with the Web, a number of associated representation technologies have been developed, such as Hyper Text Mark-up Language (HTML), eXtensible Mark-up Language (XML), and Virtual Reality Mark-up Language (VRML) to enable better cross-platform and cross-enterprise exchange of multimedia information and design models (Ane and Roller, 2011). Many early collaborative design systems were developed using the Blackboard architecture (Jagannathan, Dodhiawala, & Baum, 1989) and distributed-object technologies like CORBA (Common Object Request Broker Architecture) (Object Management Group, 2008), COM (Component Object Model) (Box, 1998), and DCOM (Distributed Component Object Model).

A blackboard architecture is a distributed computing architecture, where distributed applications modeled as intelligent agents share a common data structure, called the “blackboard”, and a scheduling/control process. The Common Object Request Broker Architecture (CORBA) is an architecture and specification for creating, distributing, and managing distributed program objects in a network. Developed by the Object Management Group (OMG), CORBA allows programs at different locations and developed by different vendors to communicate in a network through an "interface broker." The Component Object Model (COM) is an architecture and infrastructure for building fast, robust, and extensible component-based software. The COM is merely a language-independent binary-level standard defining how software components within a single address space can efficiently rendezvous and interact with each other and, at the same time, retaining a sufficient degree of separation between these components so that they can be developed and evolved independently. While, the Distributed Component Object Model (DCOM) is a seamless evolution of COM, developed by Microsoft. DCOM (Distributed Component Object Model) provides a set of concepts and program interfaces, in which client program objects can request services from server program objects on other computers in a network in a reliable, secure, and efficient manner.

Most Web-based collaborative design systems are developed using Java and CORBA (Jagannathan, Almasi, & Suvaiala, 1996; Wallis, Haag, & Foley, 1998; Huang and Mak, 1999a). Some others are developed using Common Lisp, i.e., WWDL (Zdrahal and Domingue, 1997), and Prolog, i.e., WebCADET (Caldwell and Rodgers, 1998). In addition to HTML and Java Applets, ActiveX (Huang, Lee, & Mak, 1999; Huang and Mak, 1999b) and VRML (Zdrahal and Domingue, 1997; Wallis et al., 1998) are widely used for developing client-side user interfaces.

In Figure 2, a Web-based collaborative design system on three-tier communication framework is illustrated. In this framework, the wide-area networks and the internet-based WWW infrastructure allow to develop intelligent knowledge servers. Using the knowledge-based expert system running on servers, a large-scale group of users can communicate with the system over the network (Xuan, Xue, & Ma, 2009).

Figure 2. Web-based collaborative design on three-tier communication framework (Source: Xuan, Xue, & Ma, 2009).



However, the Web technology alone is not a complete solution to collaborative design systems, although it makes remote communication physically viable through a common network. Collaborative design involves not design activity solely, but also the translation of terminology amongst disciplines as well as locating and providing distributed engineering analysis services. Thus, the Web servers should not only be a repository of information, but also provide intelligent services to assist users to solve design problems. In this regard, the application of software agents offers potential to improve the efficiency of collaborative design.

Agent-based Technology

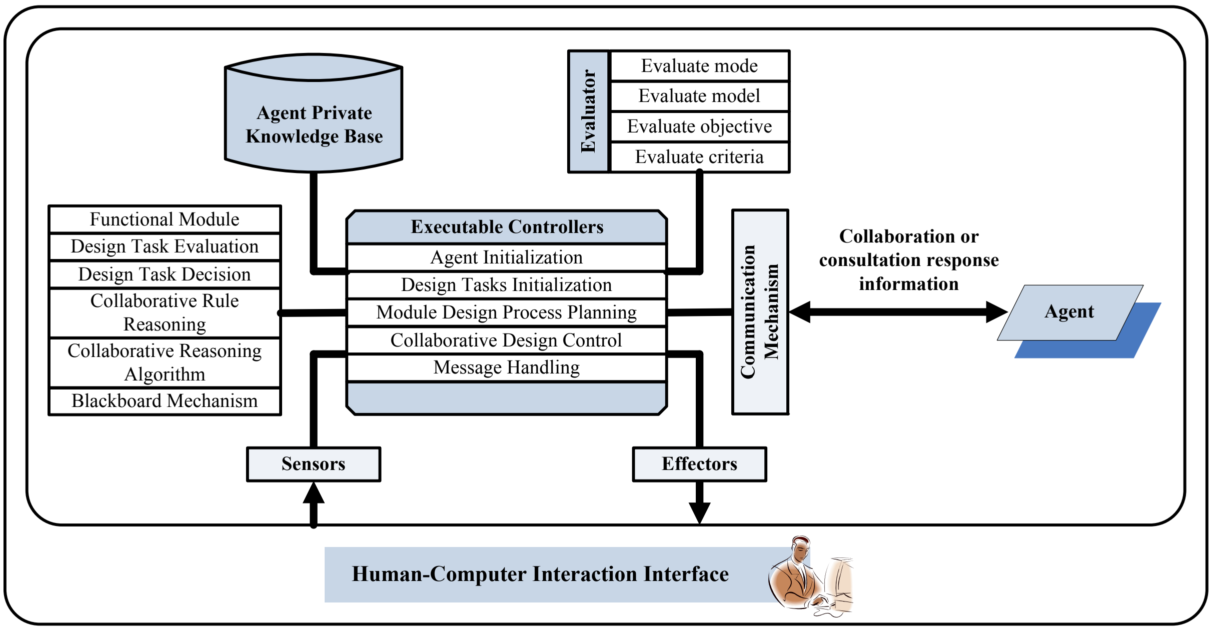
In agent-based collaborative design, software agents are mostly used for supporting cooperation amongst designers, enhancing interoperability between traditional computational tools, or allowing better distributed simulations. An agent-based collaborative design system is a loosely coupled network of problem solvers that work together to solve complex problems that are beyond their individual capabilities (Shen et al., 2008). Software agents in such systems are communicative, collaborative, autonomous, reactive (or proactive), and intelligent.

Conceptually, an agent is defined as a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objective (Weiss, 1999). *Autonomous* means that agents are able to act without the interventions of humans and other systems. They have control both over their own internal state and over their behavior.

An agent in collaborative design plans and solves the local problems of modular design by the design-related knowledge. Each agent is a knowledge system which solves the modular design problem based on their knowledge base. But the complexity of the solved problems and the size of knowledge base are less than the requirements of centralized design. Figure 3 illustrates an agent structure model for collaborative design. The main components comprise of human-computer interaction interface, sensor, effectors, executable controllers, functional module, evaluator, communication mechanism and agent private knowledge base (Ming, Jinfei, Qinghua, & Yuan, 2010).

Agents will typically sense their environment, i.e., by physical sensors or software sensors, and will have available a repertoire of actions that can be executed to modify the environment, which may appear to respond non-deterministically to the execution of these actions. An intelligent agent is one that is capable of flexible autonomous action in order to meet its design objectives. In this definition, the term of *flexibility* refers to reactivity, pro-activeness, and social ability. Reactivity represents the ability of the intelligent agents to perceive their environment, and respond in a timely fashion to changes that occur in it in order to satisfy their design objectives. Pro-activeness means that the intelligent agents are able to exhibit goal-directed behavior by taking the initiative in order to satisfy their design objectives. Social ability refers to the capability of the intelligent agent to interact with other agents (and possibly humans) in order to satisfy their design objectives.

Figure 3. Agent structure model for collaborative design (Source: Ming, Jinfei, Qinghua, & Yuan, 2010).


Agent Private Knowledge Base
Evaluator
Evaluate mode
Evaluate model
Evaluate objective
Evaluate criteria
Functional Module
Design Task Evaluation
Design Task Decision
Collaborative Rule Reasoning
Collaborative Reasoning Algorithm
Blackboard Mechanism
Executable Controllers
Agent Initialization
Design Tasks Initialization
Module Design Process Planning
Collaborative Design Control
Message Handling
Communication Mechanism
Collaboration or consultation response information
Agent
Sensors
Effectors
Human-Computer Interaction Interface

Agents operate and exist in some environment, which typically is both computational and physical. At times, the increasing interconnection and networking of computers makes such situation rare, in the usual state of affairs the agent will interact with other agents. Then, it is more convenient to deal with them collectively, as a *society of agents*. Multiagent systems are the best way to characterize or design distributed computing systems. Multiagent environments have some characteristics: (1) typically open and have no centralized designer, (2) provide an infrastructure specifying communication and interaction protocols, and (3) contain agents that are autonomous and distributed, and may be self-interested or cooperative. The rationale for interconnecting computational agents and expert systems is to enable them to cooperate in solving problems, to share expertise, to work in parallel on common problems, to be developed and implemented modularly, to be fault tolerant through redundancy, to represent multiple viewpoints and the knowledge of multiple experts, and to be reusable.

Prior research projects have demonstrated the application of software agents to collaborative design (Shen et al., 2008). PACT (Cutkosky et al., 1993) was considered as one of the earliest successful projects in this area. The interesting aspects of PACT include its federation architecture using facilitators and wrappers for legacy system integration. SHARE (Toye, Cutkosky, Leifer, Tenenbaum, & Glicksman, 1993) was concerned with developing open, heterogeneous, network-oriented environments for concurrent engineering, particularly for design information and data capturing and sharing through asynchronous communication. DIDE (Shen & Barthes, 1995) was developed to study system openness, legacy systems integration, and distributed collaboration. ICM (Fruchter, Reiner, Toye, & Leifer, 1998) developed a shared graphical modelling environment for collaborative design activities. A-Design (Campbell, Cagan, & Kotovsky, 1999) presented a new design generation methodology, which combines aspects of multi-objective optimization, multi-agent systems, and automated design synthesis. It provided designers with a new search strategy for the conceptual stages of product design that incorporates agent collaboration with an adaptive selection of design alternatives.

A successful implementation of CSCD needs a series of new strategies for efficient communication amongst multidisciplinary groups, an integration of heterogeneous software tools to realize obstacle-free engineering information exchange and sharing. In addition, a strategy is also needed for interoperability to manipulate downstream manufacturing applications as services to enable designers to evaluate manufacturability or assembleability as early as possible (Li, Ong, & Nee, 2006).

Cloud Computing

In current practice, the collaborative design systems is developed on an integrated Web- and agent-based technologies using the client/server architecture, in which the interaction and communication between agents are predefined. Due to the nature of collaborative design, the dynamic design tasks are usually involving complex and non-deterministic interactions on heterogeneous and distributed systems. This yields results that might be ambiguous and incomplete. On the other hand, designers and engineers intend to perform real-time collaborative works, where multiple users are able to share files and work on the same document simultaneously. This requirement becomes the driving force behind the use of *Cloud Computing* in collaborative design.

Baun, Kunze, Nimis, & Tai (2011) defines Cloud Computing as a technology that provides scalable, network-centric, abstracted information technology (IT) infrastructures, platforms, and applications as on-demand services, by using virtualized computing and storage resources and modern Web technologies. These services are billed on a usage basis. In this context, Cloud Computing uses virtualization and the modern Web to dynamically provide resources of various kinds as services which are provisioned electronically. These services should be available in a reliable and scalable way so that designers and engineers can use them either explicitly on-demand or simply when required.

However, this definition does not specify whether the services are provided by a distributed system or a single, high-performance server. This is in contrast to *Grid Computing* which always uses a distributed system. In general, Cloud services rely on a distributed infrastructure, but their management is typically determined in a central and proprietary manner by a single provider. This is another difference between Cloud Computing and Grid Computing where distributed nodes are usually autonomous (Wang, 2009).

The National Institute of Standards and Technology (NIST) in the U.S. (2010) specifies five essential characteristics of Cloud environment:

- **On-demand self-service:** Services can be provided unilaterally and on-demand to consumers without requiring human interaction.
- **Broad network access:** Services are available over the network in real-time through standard mechanisms.
- **Resource pooling:** The resources are pooled to enable parallel service provision to multiple users (multi-tenant model), while being adjusted to the actual demand of each user.
- **Elasticity:** Resources are rapidly provisioned in various, fine-grained quantities so that the systems can be scaled as required. To the users, the resources appear to be unlimited.
- **Measured quality of service:** The services leverage a quantitative and qualitative metering capability so that usage-based billing and validation of the service quality are possible.

Some of Cloud technologies that suitable to make effective use of the resources through different varieties of Cloud provision in the collaborative design environments are:

- **OpenNebula:** This is a fully open-source toolkit to build an 'Infrastructure as a Service' (IaaS), whether private, public or hybrid. The toolkit orchestrates storage, network, virtualization , monitoring and security technologies to enable the dynamic placement of multi-tier services on distributed infrastructures (OpenNebula, 2010). The benefits include centralized management, higher utilization of existing resources, scalability of services to meet dynamic demands and seamless integration of IT resources (Mahmood and Hill, 2011).
- **CA 3Tera AppLogic:** This is an application-centric Cloud Computing platform and a key component of Cloud solutions provided by the CA Technologies (CA Technologies, 2010). It allows for composing, running and scaling distributed applications and uses virtualization technologies to be completely compatible with existing operating systems, middleware and Web applications. The platform eliminates the binding of software and hardware through

virtualization. The applications are assembled using completely self-contained independent software components (Mahmood & Hill, 2011).

The collaborative design systems, unlike distributed systems which are interconnected over local networks, they use the distributed Cloud Computing systems that integrate heterogeneous resources, which could be located at any place on this earth where an Internet connection is available. In such environments, it is recommended to use a loosely coupled, asynchronous, message-based communication via Web services (Mahmood & Hill, 2011).

Web Service

As defined by the Web Services Architecture Working Group of the World Wide Web Consortium (W3C), a Web service is a software application identified by a uniform resource identifier (URI), whose interfaces and binding are capable of being defined, described and discovered by XML artifacts and supports direct interactions with other software applications using XML based messages via internet-based protocols (W3C, 2002). Meanwhile, Wohlstadter and Tai (2009) define Web services as distributed middleware, which enables machine-to-machine communication on the basis of Web protocols.

The Web services are determined by the intended use of technology to provide services for the specified needs. They propagate a compositional approach for application development. Functions can be integrated into an application using external or distributed services.

With regard to interoperability, the Web services describe the standards required to format and process messages as well as the standards for service interfaces. There are two approaches of interoperability. The first approach is SOAP/WSDL-based Web services. SOAP is a messaging protocol and WSDL (Web Services Description Language) is an interface description language. Thus, SOAP/WSDL-based services have programmatic interfaces. The second approach is RESTful (REpresentational State Transfer) services. REST describes a style of software architecture, which is built on top of HTTP. RESTful services can only be invoked from the uniform HTTP interface. Both SOAP/WSDL-based Web services and RESTful use uniform resource identifiers (URIs) to identify the required services (Baun et al., 2011).

MULTIAGENT-BASED PRODUCT DATA COMMUNICATION SYSTEM

In order to solve the existing problem of incompatibility that exists in collaborative design, in this section the architecture of multiagent-based product data communication system is introduced. The system is developed on a private Cloud network using the integrated Web- and agent-based technologies that enables communication and coordination amongst multiple users in the collaborative design environment.

The system architecture uses a centralized database that can be shared during the collaborative design transactions. The product database is designed on an active semantic network (ASN), i.e., a network of nodes and links, where the nodes represent objects of the real world and the links relations amongst these objects. The ASN can handle and manage the increasing amount of knowledge during the design process (Roller, Eck, Bihler, & Stolpmann, 1995; Roller & Eck, 1997; Thiel, Dalakakis, & Roller, 2005). The active component is able to make inferences and activate external actions (e.g., at a graphical user interface). Considering the complexity of engineering parts, feature taxonomy and data dictionary are built as reference in the database.

In this architecture, as it has been mentioned the data communication network is designed on the seven-layers ISO/OSI network model. The OSI model defines the communications process into seven layers, which divides the tasks involved with the moving information between networked computers into smaller and more manageable task groups. A task or group of tasks is then assigned to each of the seven OSI layers. In the following paragraphs, each component of the system is discussed.

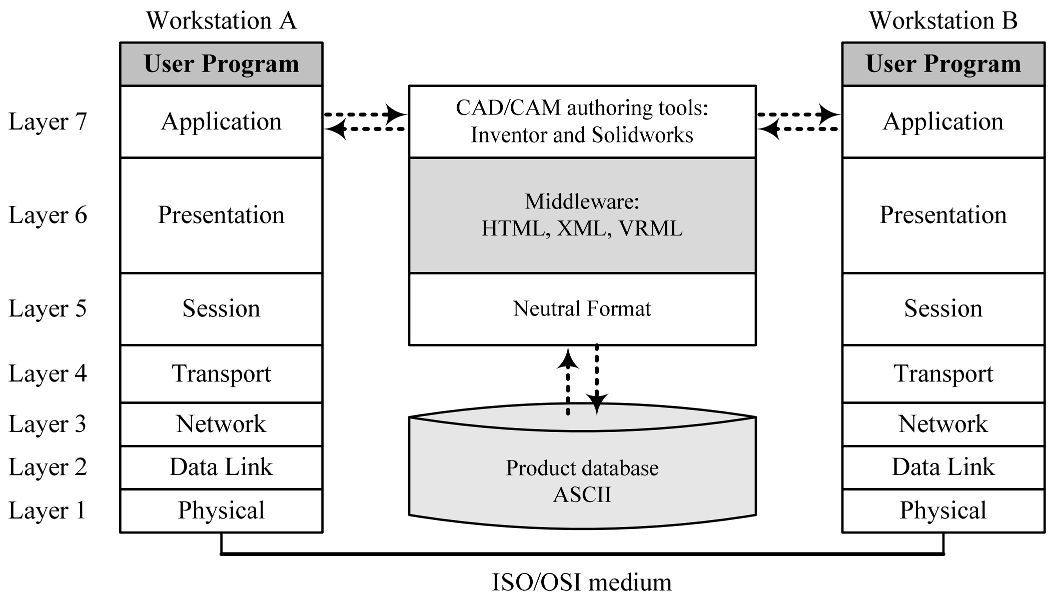
The ISO/OSI Communication Network

The seven-layers OSI model divides the communication process into seven layers, i.e., physical, data-link, network, transport, session, presentation, and application layers. Each layer is reasonably self-contained so that the tasks assigned to each layer can be implemented independently. This enables the solutions offered by one layer to be developed and replaced without adversely affecting the other layers. Figure 4 describes the ISO/OSI Product Data Communication Network.

Layers 1 to 3 deal with communications between network devices. The physical layer defines the electrical and mechanical specification of data transmission devices. This layer performs major function and service for the establishment and termination of a connection to a communications medium. It also performs modulation, or conversion, between the representation of digital data in user equipment and the corresponding signals transmitted over a communications channel. The data-link layer provides frames across a single local area network (LAN), which functions to define the resolution of contention for the use of shared transmission medium, delimitation and selection of frames addressed to given nodes, detection of noise through frame-check sequence, and any error correction or retries performed within the LAN. The network layer provides the functional and procedural means of transferring variable length data sequences from a source host on one network to a destination host on a different network, while maintaining the quality of service requested by the transport layer.

Layers 4 to 7 deal with end-to-end communications between data source and destinations. The transport layer provides transparent data transfer between end-users, providing reliable data transfer services to the upper layers by ensuring that data units are delivered error-free, in sequence, with no losses or duplications. The session layer controls the dialogues between computer systems during a communication session. This layer establishes, manages and terminates the connections between the local and remote application. It provides full-duplex operation, and establishes check-point, adjournment, termination, and restart procedures. The presentation layer establishes context between application-layer entities, in which the higher-layer entities may use different syntax and semantics if the presentation service provides a mapping between them. Finally, the application layer defines the user interface for communication process and data transfer in a network. It interacts with software applications (e.g., Autodesk Inventor, Solidworks, etc.) which implement communication components. This layer functions for determining resource availability as well as synchronizing communication process.

Figure 4. ISO/OSI Product Data Communication Network.



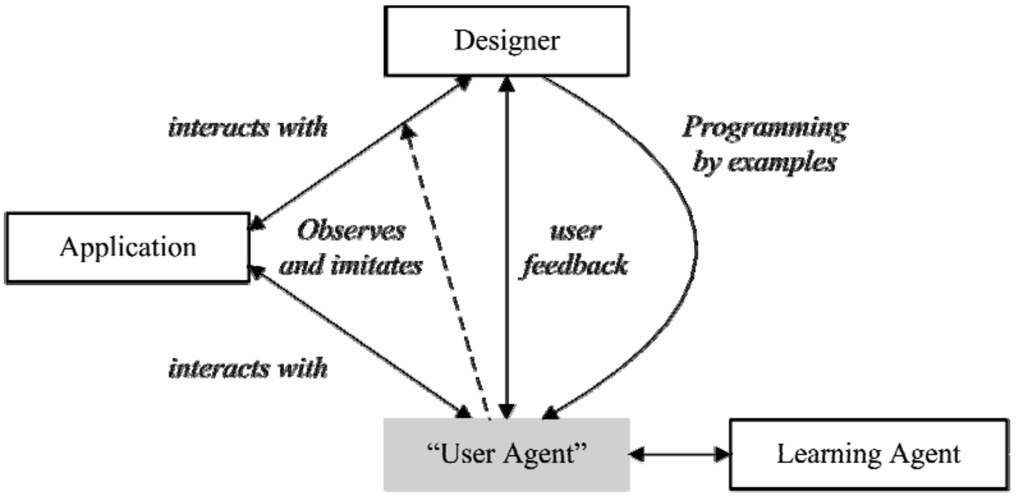
When data is sent from workstation A to workstation B, it goes down the layers. At each layer, a control message is appended to the data. Then, the complete data is transmitted through the ISO/OSI medium to workstation B. At each layer of workstation B, the control message is stripped and proper actions are taken to convert the data into a proper format. In our network model, two CAD/CAM/CAE authoring tools are implemented at the application layer, i.e., Autodesk Inventor and Solidworks. At the presentation layer, the engineering design is then communicated through a middleware on HTML, XML, and VRML formats. In this layer, design intent is further encoded/decoded into the neutral format for data storage and retrieval purposes. Finally, the design entities are encrypted/decrypted and store in the database in ASCII (American Standard Code for Information Interchange) code. The product database divides into three object-oriented databases (OOD), i.e., design, manufacturing, and process planning databases. The active OOD allows users to specify actions to be taken automatically by agents given certain rules when certain conditions arise (Roller and Eck, 1998). In the following subsection, the concept of multiagent system is discussed.

Multiagent System

On the aforementioned architecture, the product data communication system runs on an intelligent knowledge server, which embodies two capabilities of intelligent and learning agents. The intelligent agent is responsible to identify the part identity (i.e., part name, ID, date of creation, creator, model, and sub-assemblies), design (i.e., geometry, topology, dimensions, and constraints), and function based on the existing data dictionary built in the system, as well as to define similarity of a part amongst others based on its function or model. When a new design is introduced, the learning agent acts to learn the new function. Then, the agent measures similarity of the new part in comparison with the existing ones, extract related design information, create new identity and, finally, register and store it as a new entity in the data dictionary.

The aspect of learning is essential for agents. Agents are assumed to be situated in a flexible environment which they can modify while pursuing their own goals. In this environment, the agents are proactive in that they may initiate certain goal-oriented behavior. A learning agent can easily adapt with a changing environment by extending and modifying its behavior according to its own experience, i.e, by changing existing rules or adding new rules (Borghoff & Schlichter, 2000). Figure 5 describes learning triggered by a user agent. The user agent learns either by interacting with other agents, or by communicating with the users (e.g., designers, engineers, managers, etc.). The learning process can be based on user feedback, explicit user programming by examples, or by observing user interactions with the applications.

Figure 5. Learning of a user agent (Source: Borghoff & Schlichter, 2000).



The product data communication system is developed as an adaptive system. Here, the term of adaptive represents the ability of agents to adapt with changes in environment that commonly source from the changing of product database, application programs, data structure and formats, in such a manner in order to improve the system's future performance (Ane & Roller, 2011). Based on the ability of the learning agent to modify or add rules, the design knowledge can be up-dated on regular basis, thus, they are reusable. Since the design knowledge is developed on an active database system, hence, in the following subsection the active semantic network is going to be discussed.

The Active Semantic Network

The Active Semantic Network (ASN) is a shared database system developed for supporting designers during product development and is realized as an active, object-oriented database system. The ASN uses mechanisms of distributed database systems to distribute design objects in a heterogeneous environment and to allow distributed access to shared data. The goal of the ASN is to represent all knowledge relevant to the product development process and to support geographically distributed product design teams (Roller, Eck, Bihler, & Stolpmann, 1995).

Concepts of cooperative transaction models allow a group-oriented access to shared objects by multiple users (Roller, Bihler, & Eck, 1997). The objects in the database consist of three parts: the data itself, a set of associated rules, and locking objects for realizing the cooperative transaction system. Cooperative access to shared objects is realized by extended locking mechanisms which use knowledge about users and user groups in the knowledge base to support collaborative works.

Figure 6 depicts the structure of the ASN. The most important parts in this structure are a meta model to specify the structure of the ASN, a database-independent programming interface, an active component for rule processing, a cooperative transaction model and a distributed object management. The active component of the ASN is able to propagate changes at one node of the semantic network across the whole network, as well as to signal inconsistencies and conflicts to the responsible users, and start external actions (e.g., at the graphical user interface).

Before a constraint can be propagated, it has to be modelled. As illustration, a basic geometric constraint problem of piston is described in Figure 7. The performance of a piston is affected by the rotation of crankshaft that is supported by the piston rod. The crankshaft is represent as a circle where the center of the circle will be the axis of the crankshaft and the circumference represents the rotation of the center of the journal. The piston rod is represented as a straight line. The piston and the top of the piston rod travel at the same vertical velocity. Representation of piston in the object-oriented product model is described in Figure 8. In this representation, there is a dependency between a piston and a rod, which is a simple constraint between different objects.

In the following step, rules are used to evaluate parameters inside the 2-dimensional geometric objects (e.g., ellipse, triangle, etc.) and define the geometric constraints. Here, constraints propagation is realized through the Event-Condition-Action (ECA) rules. According to McCarthy and Dayal (1989), the ECA rules have the following attributes:

- An *Event* triggers a rule.
- A *Condition* is a collection of queries that are evaluated when the rule is triggered.
- An *Action* is executed when the rule is triggered and its condition is satisfied.

Using the ECA rules, the geometric constraints can be modeled in the following way:

- Event is a write operation on one of the variables of a constraint.
- Condition is a collection of queries that check if the constraint is satisfied.
- Action is executed if the constraint is not satisfied. in order to satisfy it.

Figure 6. Structure of the Active Semantic Network

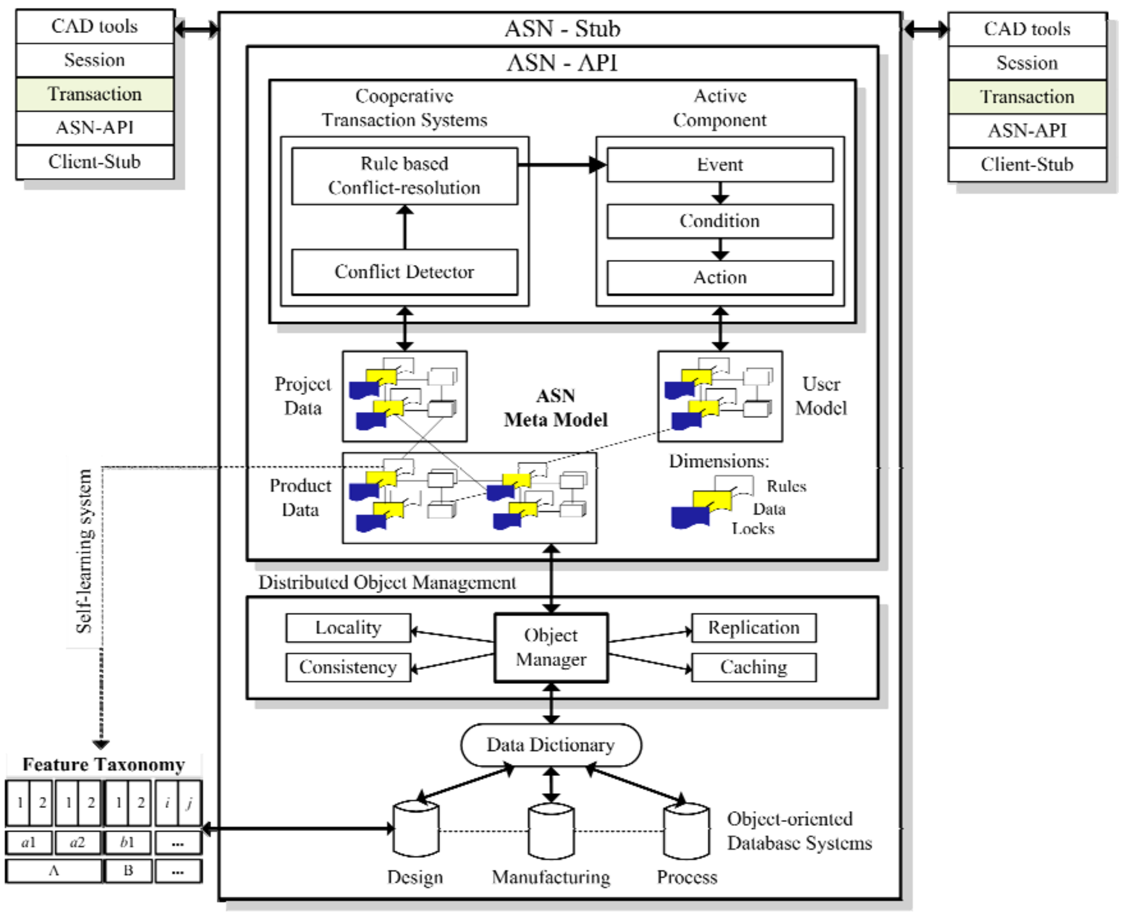
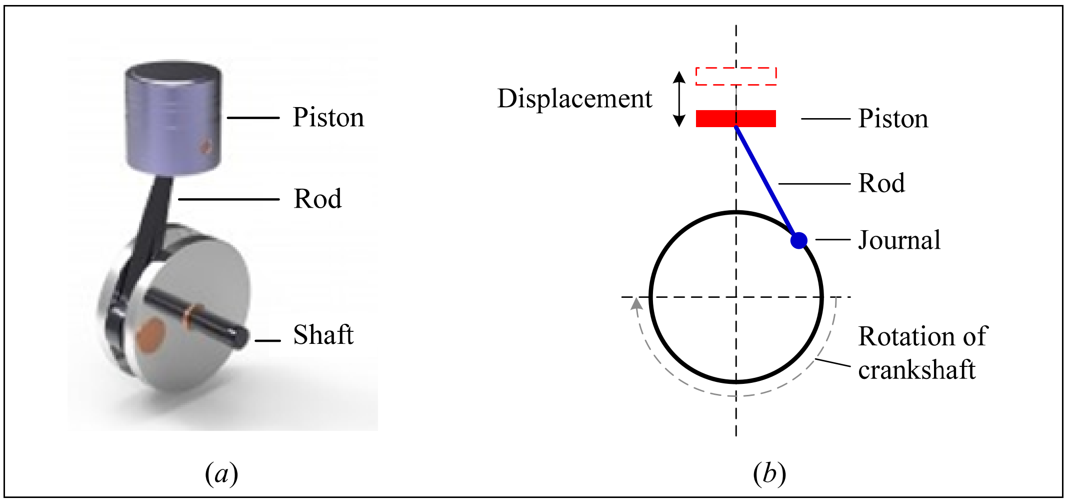
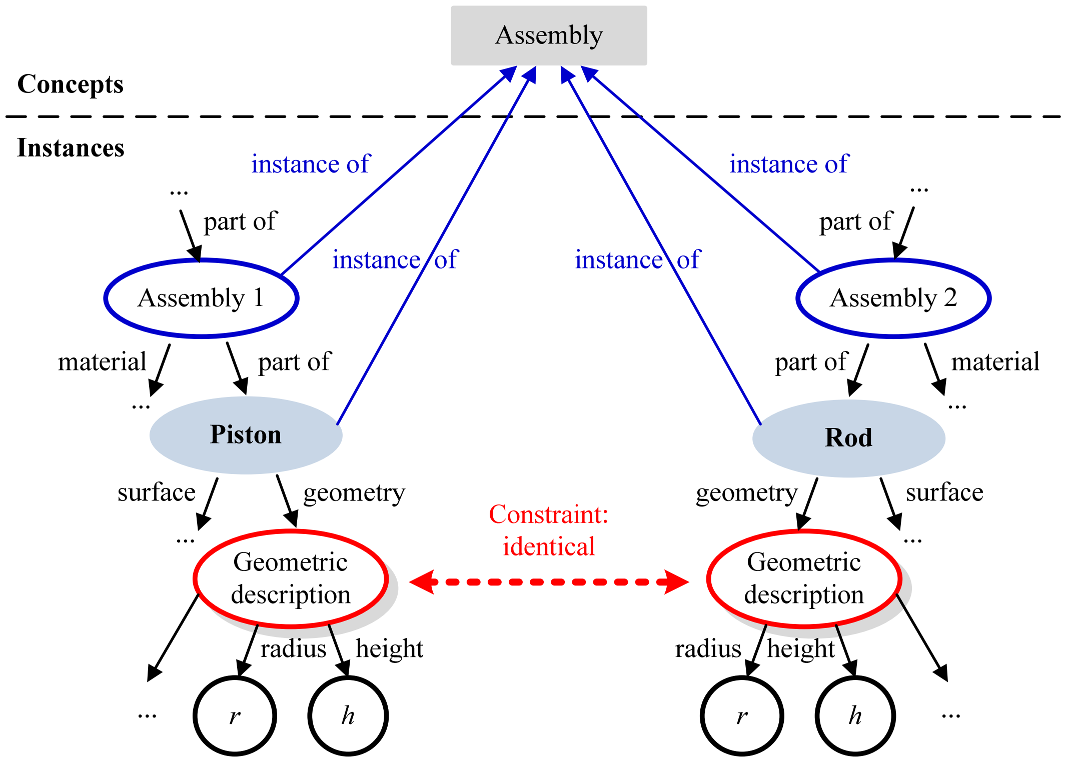


Figure 7. Piston: (*a*) product model, (*b*) definition of design problem.



Applying the ECA rules, constraints between different objects of piston can be modeled in the ASN. In constraint modeling, a new object is created which represents the geometric constraint itself. Objects in the network which model no real item in the real world but properties like ‘relationships’ or ‘constraints’, are called ”virtual objects” in the ASN. Figure 9 shows the virtual object ”Constraint =”. The constraint concept ”Identical” describes the identity of two geometry parts, and the instance of this concept models the fact that the instances ”piston” and ”rod” have the same geometry.

Figure 8. The object-oriented product model of piston.



The complete specification of all dependencies amongst all variables of a constraint is only possible, if the constraint describes in mathematical objects well represents the geometric objects. Here, the piston-crankshaft mechanism can be abstracted as a geometric constraint solving problem comprising five points p_i, $1 \leq i \leq 5$, and a straight line l_i. Figure 10 depicts geometric constraint for the piston-crankshaft mechanism (Cundy & Rollet, 1961; Hoffmann & Joan-Arinyo, 2005).

Figure 9. Representation of constraints in Active Semantic Network.

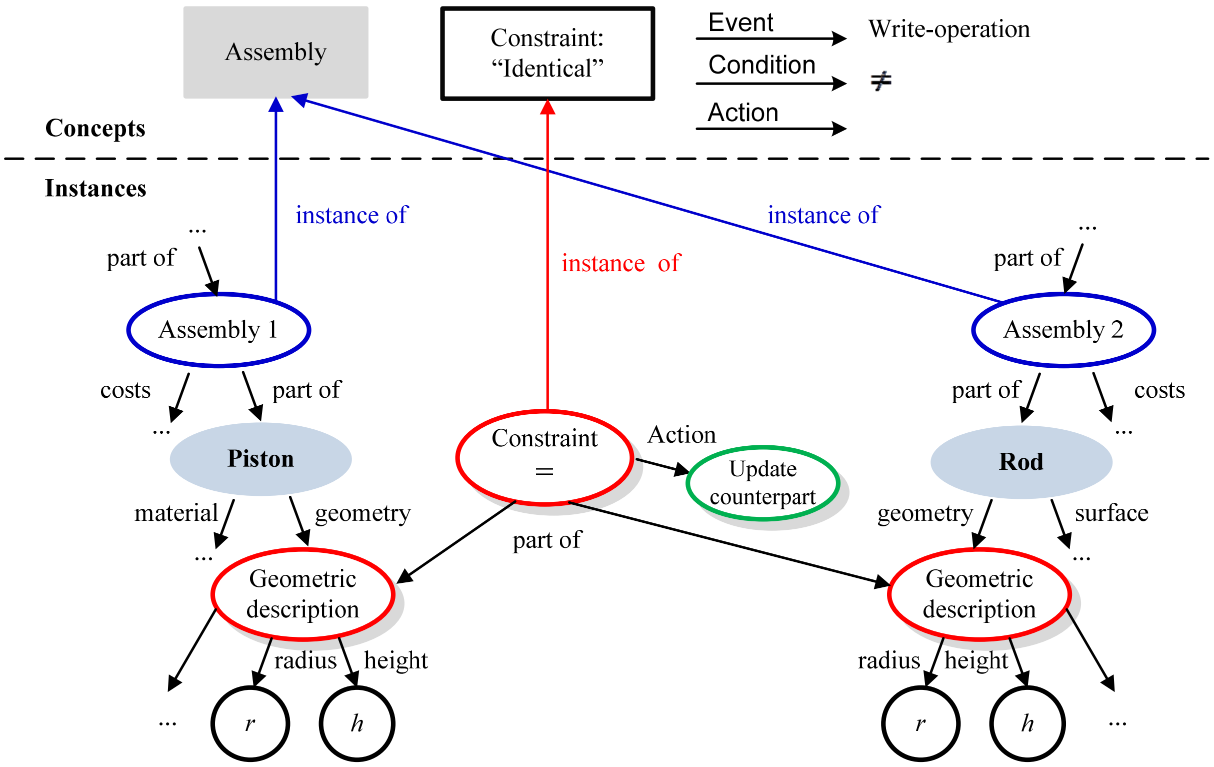
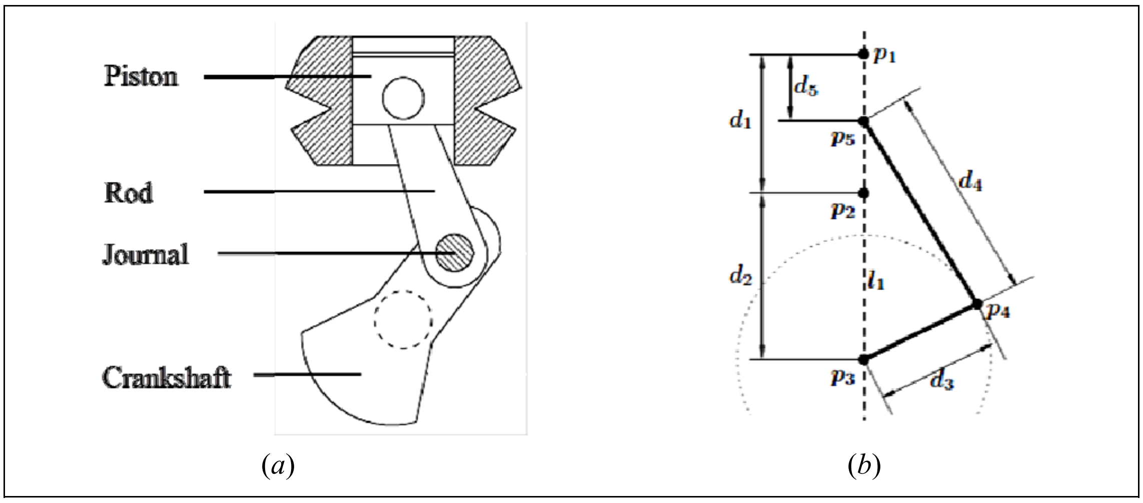


Figure 10. Piston and crankshaft mechanism: (*a*) actual mechanism, (*b*) geometric problem (Source: Hoffmann & Joan-Arinyo, 2005).

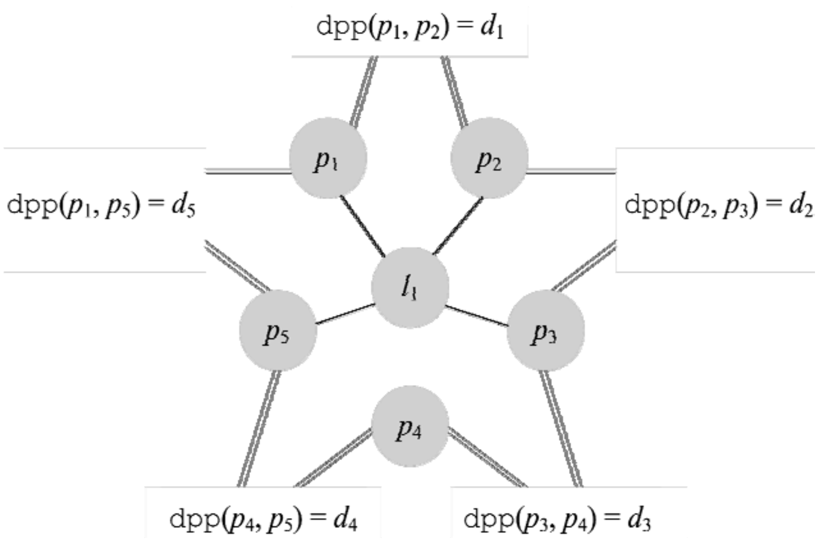


In the following process, a set of constraints which includes point-to-point distances, dpp(.), and coincidences, on(.), can be defined:

$$\begin{aligned} \mathrm{dpp}(p_1, p_2) &= d_1 & \qquad \mathrm{on}(p_1, l_1) \\ \mathrm{dpp}(p_2, p_3) &= d_2 & \qquad \mathrm{on}(p_2, l_1) \\ \mathrm{dpp}(p_3, p_4) &= d_3 & \qquad \mathrm{on}(p_3, l_1) \\ \mathrm{dpp}(p_4, p_5) &= d_4 & \qquad \mathrm{on}(p_5, l_1) \\ \mathrm{dpp}(p_1, p_5) &= d_5 & \end{aligned}$$

Based on given constraints, a constraint graph is developed to study the propagation of a constraint across the network as depicted in Figure 11. Furthermore, the constraint graph can also be used to analyze whether a system of equations is under-, well- or over-constrained from the system structure.

Figure 11. Constraint graph of piston-crankshaft mechanism.



dpp(p_1, p_2) = d_1
dpp(p_1, p_5) = d_5
dpp(p_2, p_3) = d_2
p_1
p_2
l_1
p_5
p_3
p_4
dpp(p_4, p_5) = d_4
dpp(p_3, p_4) = d_3

In the piston-crankshaft mechanism, the speed of a piston is significantly affected by the connecting rod, $s = ((d_1 - d_5) + d_2)$. For instance, if p_5 is modified from p_1, this condition affects constraint d_5 that propagates to constraints d_1 as well as the optimal distance, s. Using the Law of Cosines (Rollins, 2012), finally, the optimal s can be found:

$$b^2 = s^2 + r^2 - 2\,r\,s\cos(\theta) \qquad \text{(eq.-1)}$$

$$s = r\cos(\theta) + \sqrt{b^2 - r^2\sin(\theta)^2} \qquad \text{(eq.-2)}$$

$$s = d_3\cos(\theta) + \sqrt{d_4^2 - d_3^2\sin(\theta)^2} \qquad \text{(eq.-3)}$$

During design communication, designer needs information about product and sub-assemblies in order to establish and make transaction on the ASN. This type of information is provided in the Data Dictionary. In the following subsection, the structure of data dictionary is discussed.

Data Dictionary

A data dictionary, or also called metadata repository, is a centralized repository of information about data such as meaning, relationships to other data, origin, usage, and format (IBM Dictionary of Computing, 2008). Data dictionary is an integral part of the product database. It holds information about the database and the type of data that it stores, i.e., the metadata. It provides a collection of descriptions of the data objects or items in a data model for the benefit of designers and engineers who need to refer to them.

In the proposed architecture, data dictionary is not hidden from users that differ from most database management systems. However, the use and retrieval of data requires authorization in order to prevent them from improper use that might accidentally destroy the contents. The integrated data dictionary is called active since it controls access to the database and automatically updates as changes occur in the database.

For illustration, a data dictionary that adopts the electronic parts catalogue (ETK) of the BMW Group is presented. It provides interfaces for users to interact and access data related to vehicles model and object relationships of parts and sub-assemblies. It consists of record types (i.e., images and tables) created in the database by systems generated command files, tailored for each supported back-end product database management system. The 'back-end' refers to the database that resides in the intelligent knowledge server. Typically, entry of a data dictionary are: name of table, name of fields in each table, data type of each field (e.g., text, date, integer, image, etc.), length of each field, default value of each field, type of field (i.e., 'nullable' or 'not nullable'), and constraints applied to each field.

Using a descriptive name or Product-ID, an object in real world (e.g., gearshift, brake, wheel, etc.), its relationship with other objects, and the type of data (e.g., text, binary value, image, etc.) can be identified. Figure 12 shows data structure of vehicles parts in the data dictionary, in which the design of a crankshaft connecting rod can be accessed. The data dictionary opens a window started from layer 1 (i.e., 'Vehicle Parts') that provide information on engine object amongst a set of collection of vehicle parts. Afterwards, it continues respectively to open windows in layer 2 which provide information on the connecting rod amongst related sub-assemblies of an engine, layer 3 information on the engineering design and, finally, layer 4 information on the specification of connecting rod.

When a new design is introduced in the system and previously does not exist in the database, this will invoke the learning agent to study about the new entity through feature recognition. The recognition algorithm works by recognizing a new entity from its geometric shape based on specification in the feature taxonomy. In the following subsection, the structure of feature taxonomy is described.

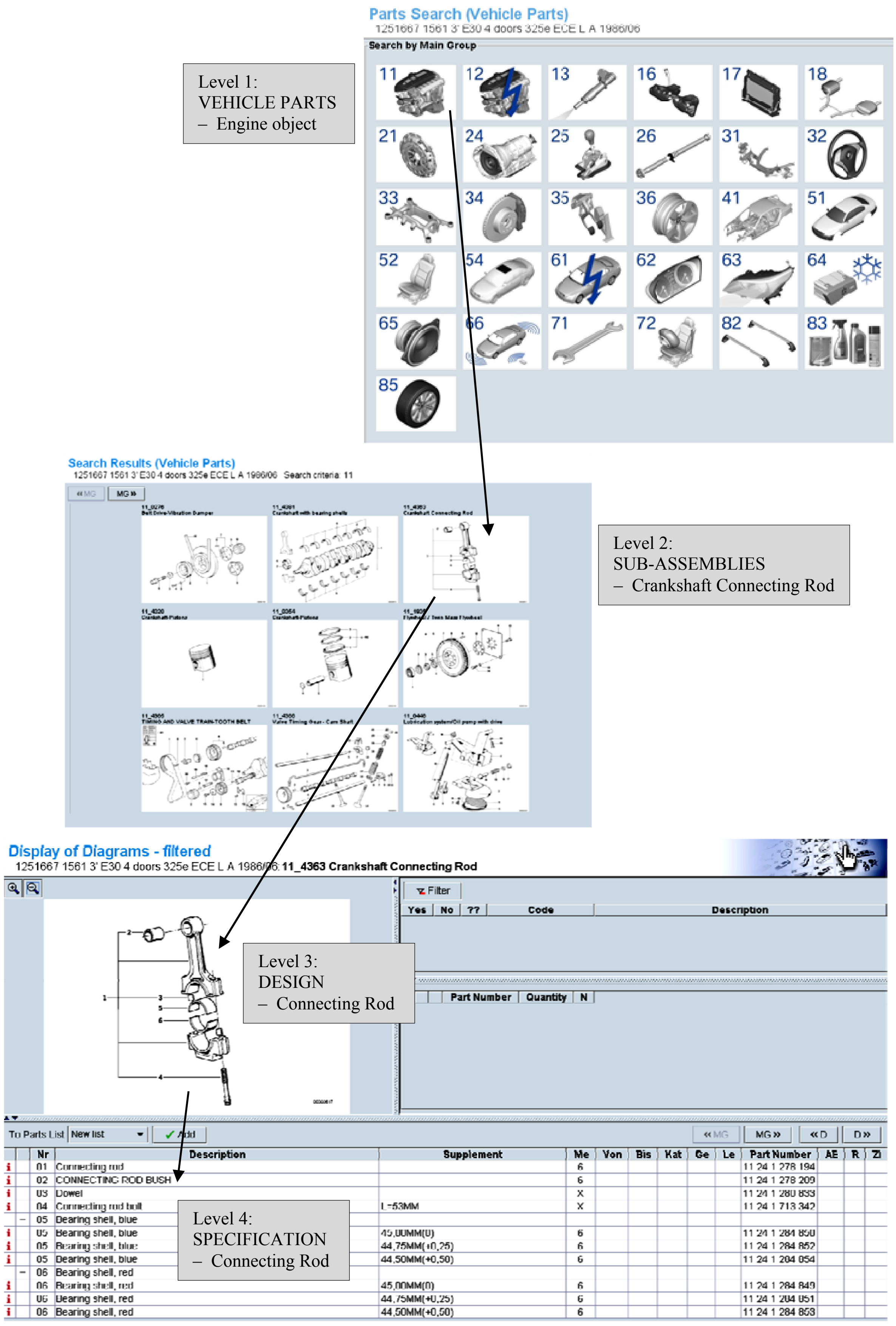
Feature Taxonomy

Generally speaking, feature means the generic shapes or characteristics of a product with which designers and engineers can associate certain attributes and knowledge useful for reasoning about given product. Many of the feature definitions are abstract in the sense they do not designate classes intended to be instantiated. Instead, they are present to give a clear structure to the feature taxonomy and to take benefit of inheritance. Therefore, it is important to outlining the higher-levels of feature taxonomy.

Referring to Shah & Mantyla (1995), feature taxonomy can be defined as a hierarchical classification of features used in design and manufacturing that gives a natural structure for developing the feature library, as well as to simplify and encourage the extension of the library. In this context, the taxonomy is oriented to feature-based manufacturing. The outline of the feature taxonomy is depicted in Figure 13. It is initiated by the root class `feature` that has direct subclasses `basic-feature`, `transition`, `billet`, `container-feature`, and `surface`. These abstract classes form the roots of subtaxonomies of features for the following purposes:

`basic-feature`	This class consists of regular features appearing as detail features in the feature models, i.e., texture, hole, rotational-feature, and prismatic-feature.
`transition`	This class consists of regular features corresponding with various types of roundings and blends between two regular features.
`billet`	This class is the root of features representing various kinds of "base objects" that form the basis of feature-based designs and also act as initial workpieces for machining applications. Typical billets include `block,` `L-block`, and more complex solids such as special castings.
`container-feature`	This class is the root of all feature types that are made of other simpler features, such as compounds, patterns, and also whole parts and assemblies.
`surface`	This includes surface types. Surfaces are used to denote areas of features for recording feature containment relationship. Surfaces are also useful to model certain manufacturing processes applied to a planar or cylindrical area.

Figure 12. Structure of data dictionary (Adapted from ETK - BMW Group, 2010).


Parts Search (Vehicle Parts)
1251667 1561 3' E30 4 doors 325e ECE L A 1986/06
Search by Main Group
11
12
13
16
17
18
21
24
25
26
31
32
33
34
35
36
41
51
52
54
61
62
63
64
65
66
71
72
82
83
85
Level 1:
VEHICLE PARTS
– Engine object
Search Results (Vehicle Parts)
1251667 1561 3' E30 4 doors 325e ECE L A 1986/06 Search criteria: 11
« MG
MG »
11_0276 Belt Drive-Vibration Damper
11_4381 Crankshaft with bearing shells
11_4363 Crankshaft Connecting Rod
11_4320 Crankshaft-Pistons
11_0354 Crankshaft-Pistons
11_1935 Flywheel / Twin Mass Flywheel
11_4366 TIMING AND VALVE TRAIN-TOOTH BELT
11_4368 Valve Timing Gear - Cam Shaft
11_0446 Lubrication system/Oil pump with drive
Level 2:
SUB-ASSEMBLIES
– Crankshaft Connecting Rod
Display of Diagrams - filtered
1251667 1561 3' E30 4 doors 325e ECE L A 1986/06 11_4363 Crankshaft Connecting Rod
Filter
Yes
No
??
Code
Description
Part Number
Quantity
N
Level 3:
DESIGN
– Connecting Rod
To Parts List
New list
Add
« MG
MG »
« D
D »
Level 4:
SPECIFICATION
– Connecting Rod

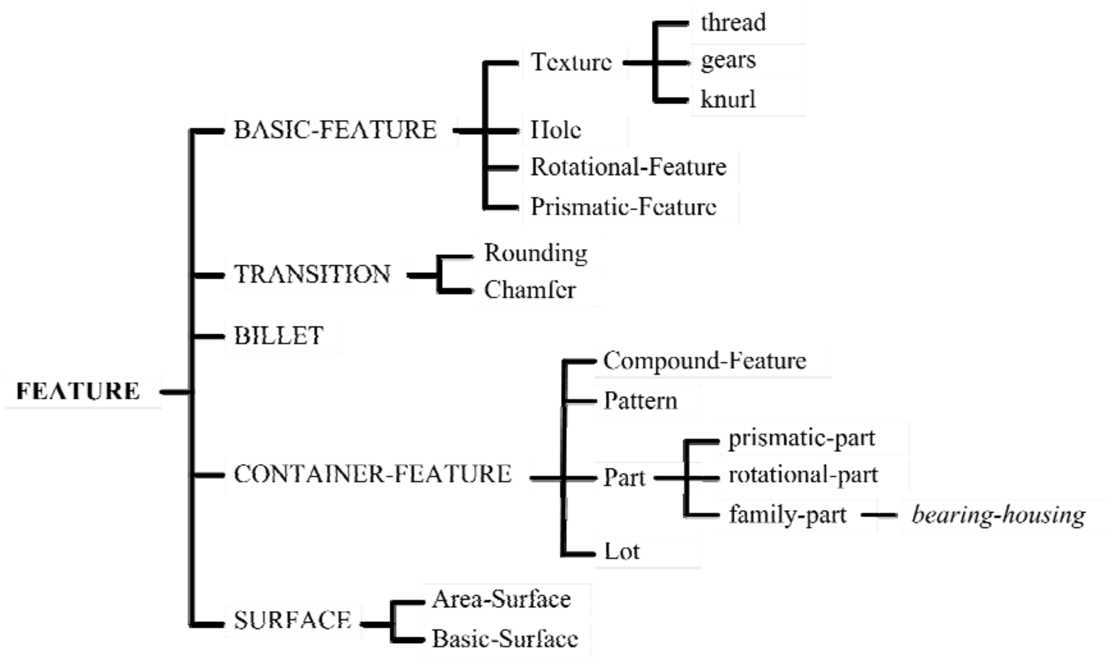
		Nr	Description	Supplement	Me	Von	Bis	Kat	Ge	Le	Part Number	AE	R	Zi
i		01	Connecting rod		6						11 24 1 278 194			
i		02	CONNECTING ROD BUSH		6						11 24 1 278 209			
i		03	Dowel		X						11 24 1 280 833			
i		04	Connecting rod bolt	L=53MM	X						11 24 1 713 342			
	–	05	Bearing shell, blue											
i		05	Bearing shell, blue	45,00MM(0)	6						11 24 1 284 850			
i		05	Bearing shell, blue	44,75MM(+0,25)	6						11 24 1 284 852			
i		05	Bearing shell, blue	44,50MM(+0,50)	6						11 24 1 284 854			
	–	06	Bearing shell, red											
i		06	Bearing shell, red	45,00MM(0)	6						11 24 1 284 849			
i		06	Bearing shell, red	44,75MM(+0,25)	6						11 24 1 284 851			
i		06	Bearing shell, red	44,50MM(+0,50)	6						11 24 1 284 853			

The definition of abstract feature classes can be described as follow:

```
        (defframe feature
          (instance-slots (position (0 0 0)) (orientation (0 0 0)))
        )
        (defframe basic-feature
          (is-a feature)
          (sign NEGATIVE)
          (parent-surface-def (constraints (axis_normal_c)))
        )
        (defframe transition
          (is-a feature)
          (instance-slots the-surfaces)
        )
        (defframe billet
          (is-a feature)
          (sign POSTIVE)
        )
        (defframe container-feature
          (is-a feature)
        )
        (defframe surface
          (is-a feature)
          (instance-slots working-allowance)
        )
```

The definition of the above subclasses defines the instance-slots `position` and `orientation` in the class `feature`. This means, all features subclasses will also contain instance-slots `position` and `orientation`. These definitions give the *x*, *y*, and *z* translations and rotation angles about *x*-, *y*-, and *z*-axes of the feature with respect to the part coordinate system. The class slot `sign` of the frame `basic-feature` states that by default, all basic features are considered ‘negative’ in that they denote removal of material from a basic part. In contrary, all billets are marked ‘positive’. The slot `parent-surface-def` states that by default, basic features are oriented along the ‘normal direction’ of the surface they are contained in. Finally, it is defined that all surfaces will contain a ‘working allowance’.

Figure 13. Feature taxonomy (Source: Shah & Mantyla, 1995).

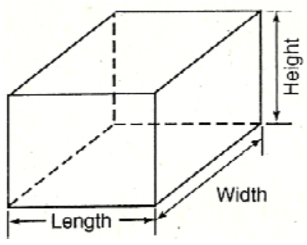
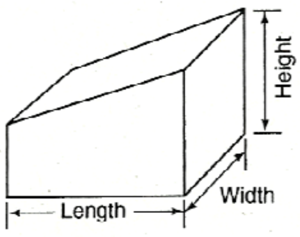
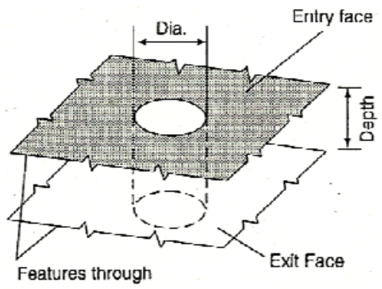
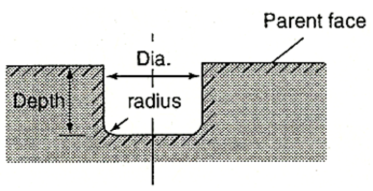


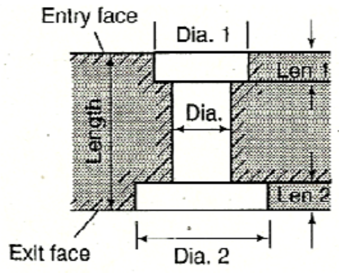
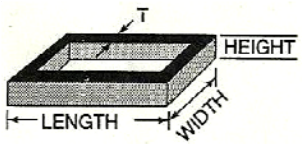
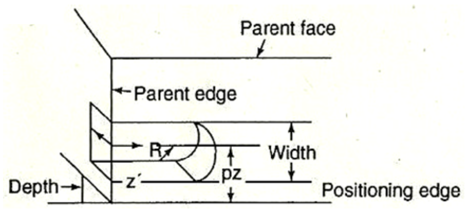
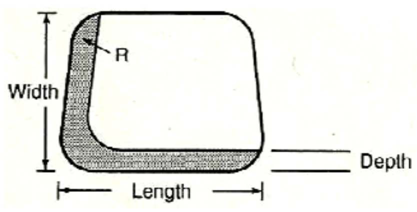
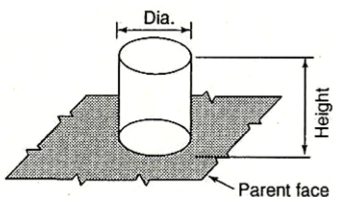
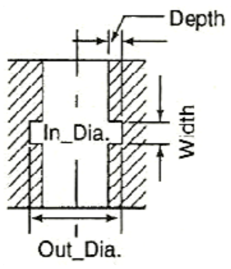
The features classes and subclasses in the feature taxonomy are implemented in conjunction with the feature library in the product database management system for feature creation and feature recognition purposes. A feature library contains generic definitions of feature classes. This might include a list of faces, which is needed to create or recognize a feature and the relationship between them. Table 1 depicts a generic feature library. Each feature is defined by its generic shape as described by the parameterized sketch. The parameters are organized into three groups: independent dimension, derived dimensions, and parameters needed in positioning the feature with respect to other features.

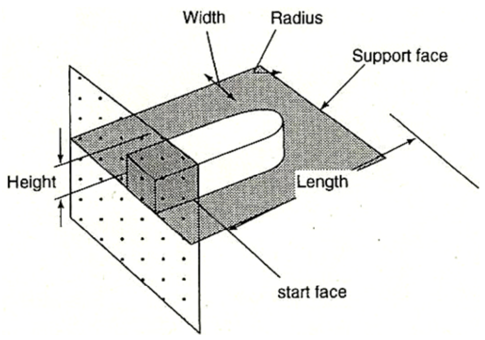
In feature recognition, the geometric model is compared to pre-defined generic features in the feature library in order to identify instances that match the pre-defined ones. Specific tasks in feature recognition might include the identification of feature geometry (i.e., edge/face growing, closure, etc.), extraction of recognized features from the geometric model (i.e., removing the portion of model associated with the recognized feature), and combination of simple features to get higher-level features.

The above subsections have described the components and structure of the multiagent-based product data communication system. Therefore, in the following section we would like to discuss the communication mechanism of the system.

Tabel 1. Generic feature library (Source: Shah & Mantyla, 1995).

Feature and Sketch	Independent Dimensions	Derived Dimensions	Positioning
Block 	Length, width, height	None	World coordinates or on the face of another block
Wedge 	Length, width, height 1, height 2	None (optional; length and width derived from parent block)	World coordinates or on the face of another block
Thru-Hole  Dia. Entry face Depth Features through Exit Face	Diameter	Depth (distance between entry and exit faces)	On entry face; specify *x*, *y* distance from reference edges on entry face
Blind Hole 	Diameter, depth, radius	None	Same as thru-hole

Feature and Sketch	Independent Dimensions	Derived Dimensions	Positioning
Double C-bore Hole  Entry face Dia. 1 Len 1 Dia. Length Len 2 Exit face Dia. 2	Dia, dia.l, dia.2, len.1, len.2	Length (optional; fixed ratios for dia.1/dia and dia.2/dia)	Same as thru-hole
Base 	Height, width, length, T		Positioned in WCS
Blind Slot  Parent face Parent edge R Width z' pz Depth Positioning edge	Depth, width, length	Radius = width/2	On planar face; along an edge, specify distance from the reference vertex
Taper Pocket (Thru) 	Length, width, R, taper_angle	Depoth (determined; by distance between entry and exit faces)	On entry face; specify *x*, *y* position of centerpoint / axes *wrt* edges (optional; specify orientation)
Boss 	Dia., height		On entry face; specify *x*, *y* position of centerpoint *wrt* edges
Recess 	Depth, width	in_dia (= hole_dia) out_dia (= in_dia + 2*depth)	Distance from entry face of hole

Feature and Sketch	Independent Dimensions	Derived Dimensions	Positioning
Rib  Width Radius Support face Height Length start face	Height, length, width	Radius (= 0.5 width)	On support face and start face as shown specify distance from reference edge

MECHANISM FOR COMMUNICATION

In collaborative design environment, a complex Web of interaction and communication is to occur. This circumstance has significant implication in the design of interaction for the multiagent-based communication system.

Synchronous Bidirectional Communication

In the aforementioned product data communication system, the interaction is designed as bidirectional communication that supports information flow from sender to receiver, and vice versa. Typically, this provokes a 'request-answer' scheme as the pattern for the data exchange. The request consists of the name of a requested service operation together with the needed parameters. The answer contains the result the receiver has obtained by executing the requested service operation using the submitted parameters.

In this design, the data exchange is synchronous. The process blocks a sender until the receiver has effectively received the message. In analogy, the receiver is blocked until the message is stored into the receiver's buffer.

Any communication between a sender and a receiver is subject to losses of request messages, losses of answer messages as well as crashes of the sending or receiving sites. In order to detect and handle the unexpected events, the product data communication system uses an 'exactly-once' call semantics, i.e., the requested service operation is processed exactly once. If repeatedly sent requests exist by a sender, at the receiving site the receiver keeps a list of current requests. Each request in the list is tagged with a unique identifier. Requests are not deleted from the list until the sender has acknowledged the correct reception of the result submitted by the receiver. Repeatedly sent requests from a sender for the same service operation are answered with the result of the first successful service operation. In any case, the receiving site does not process the same request twice. Figure 14 illustrates a communication process between a sender and a receiver, where the reception of the third request results in answering with the result obtained from processing the second request.

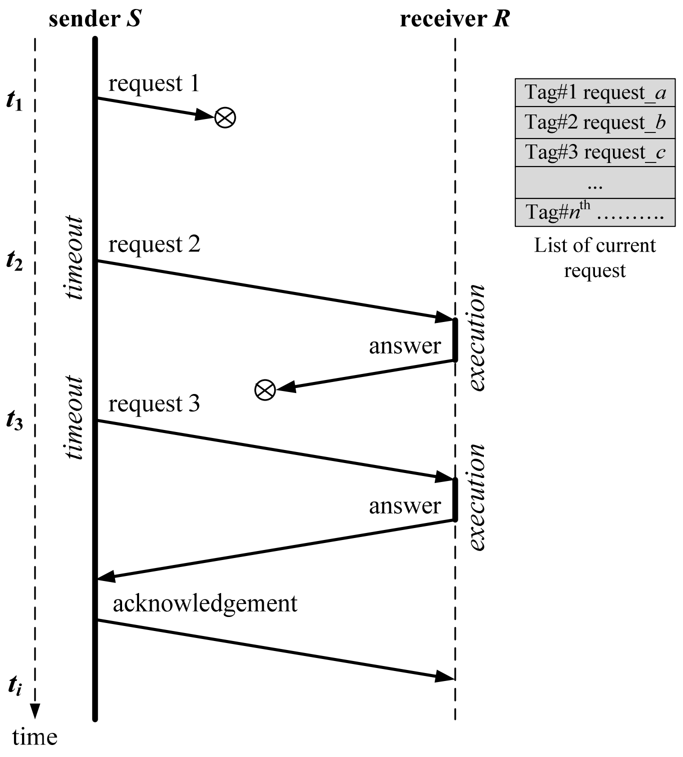
However, a failure of the receiver might occur that leads to an infinite blocking of the sender. Sender and receiver might end up in a cyclical blocking state and have to deal with the problem of deadlocks. To remedy this drawback, some sort of decoupling of sender and receiver must be performed. This decoupling can be achieved by using lightweight processes, or also called 'threads' (Borghoff & Schlichter, 2000).

Assuming a sender has invoked an operation `send` (message *M*) `to` receiver *R*, then the receiver invokes an operation of the form `receive` (message *M*, sender *S*, buffer *B*). When using remote-invocation `send`, a sender suspends execution until the receiver has received and processed a submitted request that was delivered as part of the message. In the receiver site, the reception of a message is performed in a conditional case. In the conditional reception, the resumption of execution by the receiver is dependent upon the existence of an arriving message. A precise description of this situation is described n the following algorithm,

```
Code fragment (Operation receive)
function receive(message M, sender S, buffer B): errorcode;
   if (∃ message M of sender S) then
        copy message M into buffer B;
        return true
   else return false;
```

If the expected message of a sender has arrived, the function `receive` returns `true`, otherwise `false`.

Figure 14. A request-answer scheme under an exactly-once semantics (Source: Borghoff & Schlichter, 2000).



The sender and receiver must have a similar understanding on the agent communication language being used. In the multiagent systems environment, an agent must represent its knowledge in the vocabulary of a specified ontology. All agents that share the same ontology for knowledge representation have a common understanding of the “words” in the agent communication language. In this regard, the agents need a shared ontology to be able to communicate meaningful. Thus, an ontology should be created and be accessible to the agents who are communicating.

Ontology

An ontology is a representation of knowledge of some part of the world. Ontologies provide a natural, declarative way of identifying concepts and terms that can serve as the basis for communications. Generally, an ontology is defined as a specification of the objects, concepts, and relationships in the area of interest.

Product ontology classifies the design objects into *parts*, *features*, *requirements*, and *constraints*. Axiom is used to describe the constraints and relationships amongst the objects. In this context, the design activity is a process of constructing the objects and axioms in the ontology as well as evaluating the satisfaction of requirements and constraints by the product structure and parameter values.

In the product ontology, each object is associated with a unique name that can be thought of as its ID. There are two types of objects, i.e., *class* objects and *instance* objects. A class is used for representing a generalized type or category of object, and instance for a specific member of a class. An instance and a class are related by the predicate `instanceOf(.,.)`. Afterwards, each of these classes can be further divided into subclasses. The subclass relationships are denoted by the predicate `subclassOf(.,.)`.

Given an example of `Torus-Primitive`. The term `Torus` represents a concept and the term `On` represents a relationship. Concepts can be represented in the first-order logic as unary predicates. While, the higher-arity predicates represent relationships. The idea of a torus is a primitive object can be described in the following first-order logic expression:

```
∀x(Torus x)⟹(PrimitiveObject x)
```

There are other more general representations. Instead of (`Torus T`), the expression (`instanceOf T Torus`) can be used. Both `T` and `Torus` are now objects in the universe of discourse, and relationship `instanceOf` and `subclassOf` can be introduced:

```
(class Torus)
(class PrimitiveObject)
(subclassOf Torus PrimitiveObject)
∀x,y,z (instanceOf xy) ∧ (subclassOf yz) ⟹ (instanceOf xz)
```

The last statement is a rule that expresses the notion of a type hierarchy. Looking at the product hierarchy, if two agents agree on the upper nodes of a taxonomy, they can jointly traverse the taxonomy until they find the location of a newly introduces concept. Thus, they can build a shared understanding of their content language.

In the product hierarchy, a ‘part’ is a component of the artifact being designed. The artifact itself is also viewed as a part. The structure of a part is defined in terms of the hierarchy of its component parts. The relationship between a part and its components is captured by the predicate `component_of`, which can be defined by the following axioms (Lin, Fox, & Bilgic, 1996):

Axiom 1 : Between two parts `p` and `p'`, relationship of `p` is a component (subpart) of `p'` can be expressed in:
`component_of(p, p')`

Axiom 2 : The relation `component_of` is transitive, i.e., if a part is a component of another part that is a component of a third part, then, the first part is a component of the third part:

```
(∀p1,p2,p3) component_of(p1,p2) ∧ component_of(p2,p3)
  ⊃ component_of(p1,p3).
```

Axiom 3 : The relation `component_of` is non-reflexive and anti-symmetric. This means a part cannot be a component of itself, and it is never the case that a part is a component of another part which in turn is a component of the first part:

```
(∀p)¬ component_of(p,p),
(∀p1,p2) component_of (p1,p2)⊃¬ component_of(p2,p1).
```

Axiom 4 : A part can be a subcomponent of another part. However, each part has a unique ID (i.e., its name), then, it cannot be a subcomponent of two or more distinct parts which are not components of each other:

```
(∀p1,p2,p3) component_of(p1,p2) ∧ component_of(p1,p3)
  ⊃ p2 = p3 ∨ component_of(p2,p3)∨ component_of(p3,p2).
```

Parts can be made from the same model and be identical copies, and can be used as different assemblies.

Axiom 5 : `Part1` and `Part 2` are identical copies of `Part` that treated as different instances of the same class and associated with different ID:
`instanceOf(Part1, Part),`
`instanceOf(Part2, Part).`

Parts are classified into two types, depending upon the `component_of` relationship it has with the other parts in the hierarchy. The two types are primitive and composite.

Axiom 6 : A primitive part is a part that cannot be further subdivided into components. This type of parts exist at the lowest level of the artifact decomposition hierarchy. Therefore, a primitive part cannot have subcomponents:

$$(\forall p)\ \mathtt{primitive}(p) \equiv \neg\ (\exists p')\ \mathtt{component\_of}(p',p).$$

Axiom 7 : A composite part is a composition of one or more other parts. A composite part cannot be a leaf node in the part hierarchy; thus any part that is composite is not primitive.

$$(\forall p)\ \mathtt{composite}(p) \equiv \neg\ \mathtt{primitive}(p).$$

Axiom 8 : Most composite parts are assemblies that are composed of at least two or more parts:

$$(\forall p)\ \mathtt{assembly}(p) \equiv (\exists p1,p2)\ \mathtt{component\_of}(p1,p) \wedge \mathtt{component\_of}(p2,p) \wedge p1 \neq p2.$$

Sometimes a designer may need to find out the direct components of a part. A part is a direct component of another part if there is no middle part between the two in the product hierarchy.

Axiom 9 : `p1` is a direct component of `p2` if `p1` is a component of `p2` and there is no `p'` such that `p1` is a component of `p'` and `p'` is a component of `p2`:

$$(\forall p1,p2)\ \mathtt{direct\_component\_of}(p1,p2) \equiv \mathtt{component\_of}(p1,p2) \wedge\neg\ (\exists p')\ \mathtt{component\_of}(p1,p') \wedge \mathtt{component\_of}(p',p2).$$

The `component_of` relation relates objects at lower-level in the component tree to the objects at higher-level. By the relation, it is possible to traverse upward in the component tree. However, there is also a need to traverse downward in the component tree.

Axiom 10 : A relation of object that traverse downward in the component tree is defined as `component`, which is the inverse relation of `component_of`:

$$(\forall p1,p2)\ \mathtt{has\_component}(p1,p2) \equiv \mathtt{component\_of}(p2,p1).$$

A part might have different kind of features, e.g., geometrical features, functional features, assembly features, mating features, physical features, etc. Here, features refer to the geometrical features such as hole, slot, channel, groove, boss, pad, etc.. They are also called form features.

Axiom 11 : A part and its features are related by the predicate `feature_of`:
`feature_of(f,p).`

Axiom 12 : There can be a composite feature of `f'` which are composed of several sub-features, `f1`, `f2`, ..., f_i:
`subfeature_of(f1,f).`

Axiom 13: A subfeature f1 of a feature f2 of a part p is also a feature of the part:

$$(\forall f1,f2,p)\ \mathtt{subfeature\_of}(f1,f2) \wedge \mathtt{feature\_of}(f2,p) \supset \mathtt{feature\_of}(f1,p).$$

Axiom 14: The `feature_of` and `subfeature_of` have inverse relations `has_feature` and `has_subfeature`:

```
(∀f1,f2) has_feature(f1,f2) ≡ feature_of(f2,f1),
(∀f1,f2) has_subfeature(f1,f2) ≡ subfeature_of(f2,f1).
```

Design is an evolutionary process, in which changes might occur frequently during the process. Before reaching its maturity, each object of parts, features, and constraints might undergo many transformations and revisions. Therefore, versions of design need to be created as objects to record the history of designs.

By understanding the architecture and communication mechanism of the multiagent-based product data communication system, in the following subsection we provide an illustration on the capability of the multiagent-based communication system to perform tracking of design changes.

TRACKING OF DESIGN CHANGES

In collaborative design environment, designers deal with many possibilities and complexities of design changes. During design transactions, they must assess the impacts of a design change on the other design objects and notify other parties promptly. In most design environment, due to the lack of tracking of design changes, designers have to manually perform a consistency check for a proposed design change as well as to identify all the impacts of the change. This condition makes design consistency and accuracy are not guaranteed and the productivity of design team is compromised (Wang, Shen, Xie, Neelamkavil, & Pardasani, 2000).

In order to solve this drawback, an approach to the tracking of design changes in a unified framework (Xie, 2001) is introduced. This approach enables a team of designers work on a Web-based collaborative design in distributed remote locations to visualize, manipulate and evaluate the impacts of changes to a design.

Product Data Driven Approach

From the perspectives of product variation and product improvement, design changes are imperative in order to meet the requirements of product specification, safety standards, or to reduce manufacturing or maintenance costs**.** Due to the associated relationships between elements in a product, a design change might have an impact on certain geometric constraints that might propagate to other elements of the product. In this approach, the mechanism for tracking of design changes is based on product data and the relationships generated during various stages of a product design process.

The product data contains product descriptions for product specification, function decomposition structure, solution principles, layout design, assemblies, and parts. The relationships are established between two or more elements having geometric constraint relationships to one another. Here, the geometric constraint relationship defines three types of constraint between parts, i.e., fit, contact, and consistent constraints. The fit constraint exists if there is a tolerance requirement between parts. The contact constraint represents a physical contact between two parts. The consistent constraint exists if two parts hold a dimensional constraint without a physical contact.

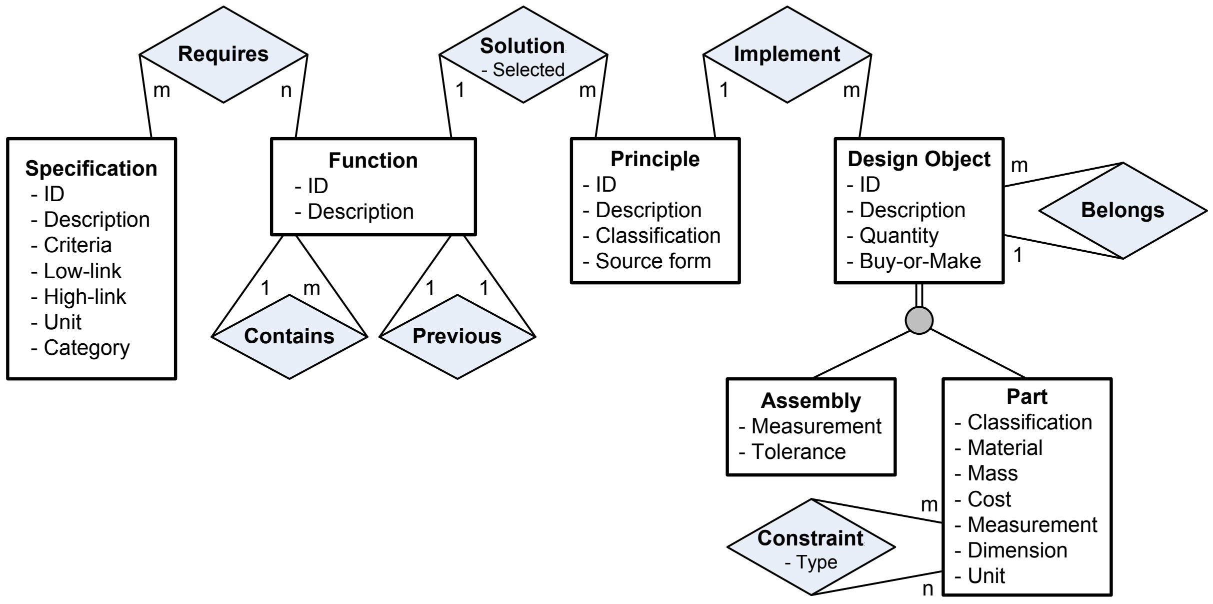
The product data driven approach enables designers to identify the total impacts of a proposed design change on an entire product. It provides a fundamental mechanism to support forward tracking and backward tracking of a design change. Forward tracking identifies the impact of the change on later design stages if a design change occurs at an earlier stage. On the other hand, backward tracking identifies the impacts of changes on previous stages, if a change occurs at a later stage. These operations can be implemented with database support.

To make the necessary design information available, product data information is extracted from various stages of design process and represented in a data model. A data model is a set of concepts that can be used to describe the structure of a database (Elmasri and Navather, 1989). In this process, an entity-relationship (E-R) model is used to describe product data with such concepts as entities, attributes, and relationships.

The entities and attributes describe an abstract object with its properties. The change tracking model includes five types of entities, i.e., *Specification*, *Function*, *Principle*, *Design_Object*, *Assembly*, and *Part*. These entities are associated to 19 attributes, i.e., *ID, Description*, *Criteria, Low-limit, High-limit*, *Unit, Category, Source-form, Selected, Quantity, Buy-or-make, Measurement, Tolerance, Classification, Materials, Mass, Cost, Dimension, and Type.*

The relationships represent a set of associations amongst entities. They limit the possible combinations of entity instances. Cardinality ratio constraints specify three common combinations for binary relationship types, i.e., *one-to-one* (1:1), *one-to-many* (1:M), and *many-to-many* (M:N). The change tracking model includes seven types of relationship, i.e., *Requires*, *Contains*, *Previous*, *Solution*, *Implement*, *Belongs*, and *Constraint*. Figure 15 describes the relationships between entities and their associated attributes. In order to provide better understanding on the change tracking model, the following subsection illustrates the forward tracking process of design changes on a motor-body.

Figure 15. E-R diagram for change tracking model (Source: Xie, 2001).



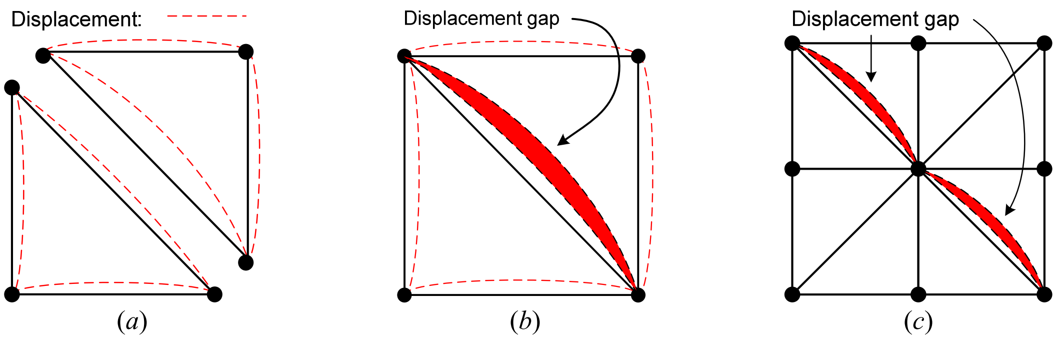
Case Study

The initial design of the motor-body is cylinder shape made from alloy steel (SS) with mass 1.751 kg and volume 0.000227 m^3. The motor-body is designed must be at a fixed position inside the vehicle engine compartment. During operation, the force produced by the electrical motor inside the motor-body is uniformly distributed from center to boundary. Therefore, three fixtures are applied at the bottom, and left- and right sides to clamp the motor-body at the fixed position.

The real structure of motor-body sustains a distributed state of stress. The stress is represented by forces at the element joints or nodes. Correspondingly, the displacement of these points is employed in the characterization of displaced state of the element.

To describe nodes of finite element in a continuum structure, they are assumed in the triangular elements that vary quadratically over the surface of the elements. When the elements are joined, there will be a general lack of continuity of the edge displacement of the elements along the juncture line as described in Figure 16. The joining of the elements at the vertices enforces the continuity of displacements only at those points. The three points of a triangular element define uniquely the shape of displacement. Since only the two end points of the element interface have influence in defining the shape of displacement along an edge, then, the displacements of the edges of the respective elements will differ for each case. As more elements are used, the disparity of the displacement of the adjacent element edges is reduced and the error in the solution due to this condition is reduced as well.

Figure 16. Displacement of elements: (*a*) deflected shapes of individual elements, (*b*) disparity of displacements along juncture line connected elements, and (*c*) reduction of displacement gap by refinement of gridwork (Source: Gallagher, 1975).



Most structural analysis problem can be treated as linear static problem under assumptions small deformation (i.e., loading pattern is not changing due to the deformed shape), elastic material (i.e., no plasticity), and static load (i.e., the load is applied to the structure in a slow of steady operation). To estimate the possible occurrence of shape deformation, the force-displacement analysis is applied. The relationship between the joint forces and the joint displacements of finite elements should satisfy the following stiffness function,

$$\{\mathrm{F}\} = [\mathrm{k}]\ \{\Delta\} \tag{eq.-4}$$

where {F}: element force, {Δ}: displacement vectors, and [*k*]: element stiffness matrix. An individual term of the [*k*] matrix, k_{ij}, is an element stiffness coefficient. If the displacement Δ_j is imposed at unit value and all other degree of freedom (d.o.f.) are held fixed against displacement ($\Delta_k = 0$, $k \neq j$), then the force F_i is equal in value to k_{ij}.

During the design optimization, the displacement analysis is performed using the mixed force-displacement equation that defines a relationship between vectors containing both force and displacement. If the forces and corresponding d.o.f. of an element are divided into two groups, designated by subscripts *s* and *f*, the general form of a mixed representation can be defined as,

$$\begin{Bmatrix} \mathrm{F}_f \\ \Delta_f \end{Bmatrix} = [\Omega] \begin{Bmatrix} \mathrm{F}_s \\ \Delta_s \end{Bmatrix} \tag{eq.-5}$$

The force-displacement analysis produces an average deformation scale at $1.7250e^{+8}$ and a prediction of location where the deformed mesh are most possible to occur. The resultants displacement shows the minimum condition 0 mm is at location (3.969 cm, -0.499 cm, -11.000 cm) and the maximum condition $6.76612e^{-8}$ mm at location (-3.373 cm, -3.291 cm, -0.099 cm). The analysis predicts that two most possible deformed locations likely to occur at the lower-part of cab-screw holes. This condition makes the cylinder shape has more possibility to slip from its position and fixtures.

Applying the stress analysis, a convenient mode of selection of stress fields that satisfy the equilibrium differential equations can be determined by using the stress functions. Stress functions are parameters which, when differentiated in accordance with certain rules, give stress components that automatically satisfy the differential equations of equilibrium.

Stresses can be evaluated at any point, both inside the element and at the nodes. For small strain and small rotations,

$$\varepsilon_x = \frac{\partial u}{\partial x}, \qquad \varepsilon_{yx} = \frac{\partial v}{\partial y}, \qquad \gamma_{xy} = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \tag{eq.-6}$$

or, in matrix form

$$\begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{Bmatrix} = \begin{bmatrix} \partial/\partial x & 0 \\ 0 & \partial/\partial y \\ \partial/\partial y & \partial/\partial x \end{bmatrix} \begin{Bmatrix} u \\ v \end{Bmatrix}. \qquad \text{(eq.-7)}$$

where u and v are nodal displacements. Based on this relationship, if the displacements functions are represented by polynomial, thus, the stresses are one-order lower than the displacements.

In this analysis, the von Mises stress function is applied (Liua et al., 2005)

$$\sigma_e = \frac{1}{\sqrt{2}}\sqrt{(\sigma_1 - \sigma_2)^2 + (\sigma_2 - \sigma_3)^2 + (\sigma_3 - \sigma_1)^2} \qquad \text{(eq.-8)}$$

where σ_1, σ_2, and σ_3 are three principle stresses at the considered point in a structure. For a ductile material, the von Mises stress (σ_e) and the yield stress of the material (σ_y) must satisfy the following constraint

$$\sigma_e \leq \sigma_Y. \qquad \text{(eq.-9)}$$

For *2*-dimensional problem, the two principle stresses in the plane are determined by

$$\sigma_1^P = \frac{\sigma_x + \sigma_y}{2} + \sqrt{\left(\frac{\sigma_x - \sigma_y}{2}\right)^2 + \tau_{xy}^2}$$

$$\sigma_2^P = \frac{\sigma_x + \sigma_y}{2} - \sqrt{\left(\frac{\sigma_x - \sigma_y}{2}\right)^2 + \tau_{xy}^2} \qquad \text{(eq.-10)}$$

Thus, the von Mises stress can be expressed in terms of the stress components in the xy coordinate system as

$$\sigma_e = \sqrt{\left(\sigma_x + \sigma_y\right)^2 - 3\left(\sigma_x\sigma_y - \tau_{xy}^2\right)}. \qquad \text{(eq.-11)}$$

The result of stress analysis estimates minimum stress 3.04976e^{-6} N/mm^2 (MPa) at location (4.170 cm, -0.470 cm, -6.750 cm) and maximum stress 0.000572322 N/mm^2 (MPa) at location (3.524 cm, -3.385 cm, -0.350 cm). The stress is distributed from the inner cylindrical mesh boundary to the outer boundary with the highest strained locations are found at the elements adjacent to the four cab-screw holes. This condition makes the initial design has high potential failure (i.e., crack) during the assembly and product use. Therefore, the initial design needs to be improved.

Based on prior design history, designers decide to change the design of motor-body from cylinder to block shape. The change of *Design_Object* from cylinder to block shape has driven direct changing of *assembly*, *part*, and *constraint* respectively. In this case, tracking of design changes is performed based on entities, attributes, and relationships, which have been determined in the E-R diagram. Changing of geometry has an impact to constraints of four cab-screw holes which, in its turn, propagate to the changing of dimensions and positions of those holes on the modified shape. The progress for forward tracking of change of *Design_Object* is described in Table 2.

As a result, the collaborative design produces an optimal design of motor-body in block shape with dimension 82.5 x 82.5 x 100 millimeters using material alloy steel (SS) with mass 1.859 kg and volume 0.000241 m^3. The force-displacement analysis of the new design generates an average deformation scale 8.59488e^{+7}, i.e., 50.18% better than the initial design. On the block shape, the deformation has been localized at the upper-front to upper-middle of finite mesh. The resultants displacement shows the minimum condition 0 mm is at location (41.25 mm, 23.25 mm, 0 mm) and the maximum condition 1.28963e^{-+7} mm at location 0.09 mm, 44.50 mm, -108.64 mm). Figure 17 and Figure 18 describe the design of motor body in comparison before and after the design optimization, as well as the results of force-displacement and von Mises stress analysis.

Table 2. Forward tracking for change of *Design_Object*.

ENTITIES	ATTRIBUTES	STATUS	DESIGN VALUES	
			INITIAL	OPTIMAL
Function	ID	No change	part#1	part#1
	Description	No change	Motor-body	Motor-body
Principle	Classification	No change	Motor protection	Motor protection
	Source Form	No change	In-house production house	In-house production
Design_Object	Quantity	No change	100 pieces	100 pieces
	Buy-or-Make	No change	make	make
Specification	Criteria	No change	mechanic – static	mechanic – static
	Low-limit	Change	3.04976e-006 N/mm^2	3.43238e-007 N/mm^2
	High-limit	Change	0.000572322 N/mm^2	0.00139487 N/mm^2
	Unit	No change	1	1
	Category	No change	automotive part	automotive part
Part	Material	No change	alloy steel (SS)	alloy steel (SS)
	Mass	No change	min 1.750 – max 1.860 kg	min 1.750 – max 1.860 kg
	Cost	No change	USD 367.82 - USD 375.00	USD 367.82 - USD 375.00
	Measurement	No change	millimeter (mm)	millimeter (mm)
	Dimension	Change	*d*: 82.5mm, *ℓ*: 110 mm	*w*: 82.5 mm, *h*: 82.5 mm, *ℓ*: 110 mm

Figure 17. Initial design: (*a*) cylinder motor-body, (*b*) 3D cross-section of deformed mesh, (*c*) estimated deformed mesh, (d) estimated strained mesh.

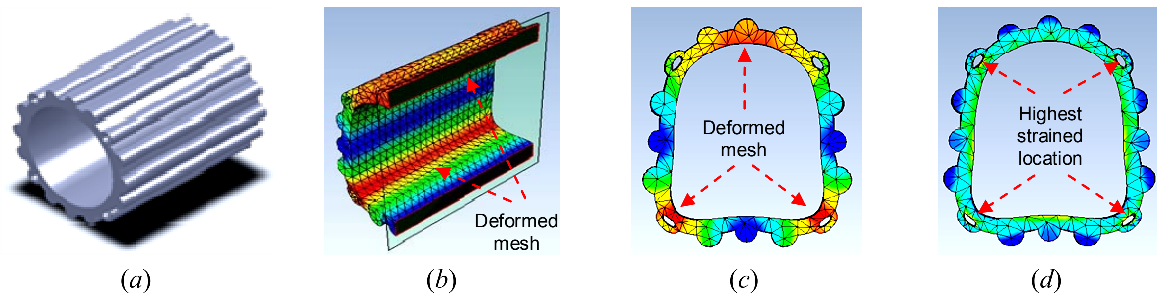
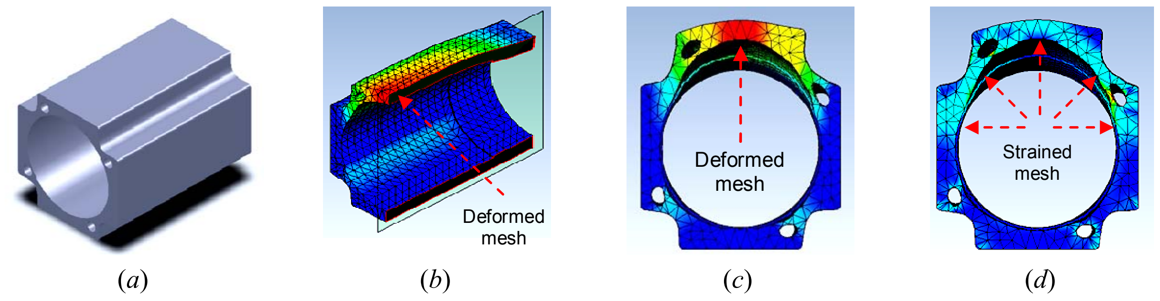


Figure 18. Optimal design: (*a*) block motor-body, (*b*) 3D cross-section of deformed mesh, (*c*) estimated deformed mesh, (d) estimated strained mesh.



CONCLUSION

Today, product development is inevitable must be performed as an integrated process with design to manufacturing. In a collaborative design environment, design activity might take place at distributed geographic locations that involves multidisciplinary design team work with various CAD/CAM application systems. This circumstance has driven a need for better communication and coordination amongst design team during the design process. As the information and communication technologies advance, the application of collaborative engineering to product design, so-called computer supported collaborative design (CSCD), becomes more promising.

In collaborative design environment, Web technology makes remote communication physically viable through a common network. Furthermore, an integrated Web- and agent-based CSCD is able to support cooperation amongst designers and engineers, enhancing interoperability between authoring tools, and allow better communication of modular design tasks. In current practice, designers and engineers intend to perform real-time collaborative works, where multiple users are able to share files and work on the same document simultaneously. With the support of Cloud technology, designers can use virtualization and the modern Web to dynamically provide resources of various kinds as services for collaborative design, which are provisioned electronically.

In this chapter, architecture of multiagent-based product data communication system has been introduced. The multiagent system runs on an intelligent knowledge server, which embodies two capabilities of intelligent agent and learning agents. It is developed as an adaptive system, which represents the ability of agents to adapt with changes in its environment in order to improve the system's future performance.

The system architecture uses a centralized database that can be shared during the collaborative design transactions. The product database is designed on an active semantic network (ASN) that can handle and manage the increasing amount of knowledge during the design process. The product database is coupled with data dictionary and feature taxonomy. Data dictionary provides a collection of descriptions of the data objects or items in a data model for users who need to refer to them during the design transaction. When a new design is introduced and previously does not exist in the database, the learning agent will be invoked to perform feature recognition. Here, the recognition algorithm works by recognizing a new entity from its geometric shape based on specification in the feature taxonomy.

The data communication network is based on the seven-layer ISO/OSI network model. During communication, the interaction is designed as bidirectional communication that supports information flow from sender to receiver, and vice versa. Since an agent must represent its knowledge in the vocabulary of a specified ontology, hence, product ontology is created that must be accessible to the agents who are communicating. Thus, all agents that share the same ontology for knowledge representation have a common understanding in the agent communication language.

In practice, design changes are imperative in order to meet the requirements of product specification, safety standards, or to reduce product costs. The product data communication system provides an ability to the tracking of design changes in a unified framework. The mechanism for tracking of design changes is based on product data and their relationships. The product data driven approach provides a fundamental mechanism to support forward tracking and backward tracking of a design change. This approach enables designers work on a Web-based collaborative design in distributed locations to visualize, manipulate and evaluate the total impacts of a proposed design change on an entire product.

ACKNOWLEDGMENT

This research project is organized under the financial support of the Institute of Computer-aided Product Development Systems, Universität Stuttgart and The Alexander von Humboldt Foundation, Germany.

REFERENCES

Ane, B. K. (2010). *Automation of design for Reverse Engineering in a collaborative design environment*: *Development of STEP neutral format translator for product data exchange*. Research Report, Stuttgart, Germany: IRIS, Universität Stuttgart.

Ane, B. K. & Roller, D. (2011). An architecture of adaptive product data communication system for collaborative design. *Proceedings of the 4th International Conference on Advances in Computer-Human Interactions* (*ACHI'11*), 182-187.

Baun, C., Kunze, M., Nimis, J., & Tai, S. (2011). *Cloud Computing*: *Web-based dynamic IT services*. Heidelberg, Germany: Springer-Verlag.

Bernd, B., Heiko, G., & Michael, W. (2007). Aktives Semantiches Konstruktions- und Zuverlässigkeitsnetz. In B. Bertsche and H.-J. Bullinger (Eds.), *Entwicklung und Erprobung Innovativer Produkte-Rapid Prototyping*: *Grundlagen, Rahmenbedingungen und Realisierung* (pp. 130-158). Berlin, Germany: Springer-Verlag.

Borghoff, U. M. & Schlichter, J. H. (2000). *Computer-Supported Cooperative Work*: *Introduction to Distributed Applications*. Heidelberg, Germany: Springer-Verlag.

Box, D. (1998). *Essential COM*. Upper Saddle River, USA: Addison-Wesley.

CA Technologies (2010). Turnkey cloud computing. Retrieved September 14, 2011 from http://www.3tera.com/AppLogic (accessed in September 2011).

Caldwell, N. H. M. & Rodgers, P. A. (1998). WebCADET: Facilitating distributed design support. *Proceedings of the IEEE Colloquium on Web-Based Knowledge Servers*, 9/1–9/4.

Campbell, M. I., Cagan, J., & Kotovsky, K. (1999). A-Design: An agent-based approach to conceptual design in a dynamic environment. *Research in Engineering Design*, *11*, 172-192.

Cundy, H. M. & Rollet, A. P. (1961). *Mathematical models*. 2nd edition, Oxford, England: Oxford University Press.

Cutkosky, M. R., Engelmore, R. S., Fikes, R. E., Genesereth, M. R., Gruber, T. R., Mark, W. S., Tenenbaum, J. M., & Weber, J. C. (1993). PACT: An experiment in integrating concurrent engineering systems. *IEEE Computer*, *26*, 28-37.

Elmasri, R. & Navather, S . B. (1989). *Fundamentals of database systems*. New York, USA: Benjamin Cummings Publishing Company Inc.

ETK - BMW Group (2010). *Electronic Parts Catalogue*. Database version: ETK 2.32, Munich, German: BMW Group.

Fruchter, R., Reiner, K. A., Toye, G., & Leifer, L. J. (1996). Collaborative mechatronic system design. *Concurrent Engineering*: *Research and Applications*, *4*, 401–412.

Gallagher, R. H. (1975). *Element analysis*: *Fundamentals*. New Jersery, USA: Prentice Hall.

Greif, I. (1988). *Computer-Supported Cooperative Work*: *A book of readings*. San Mateo, USA: Morgan Kaufmann Publishers.

Hague, M. J. & Taleb-Bendiab, A. (1998). Tool for management of concurrent conceptual engineering design. *Concurrent Engineering*: *Research and Applications*, *6*, 111-129.

Hoffmann, C. M. & Joan-Arinyo, R. (2005). *A brief on constraint solving*. West Lafayette, USA: Purdue University.

Huang, G. Q., Lee, S.W., & Mak, K. L. (1999). Web-based product and process data modelling in concurrent 'Design for X'. *Robotics and Computer-Integrated Manufacturing*, *15*, 53-63.

Huang, K., Li, X., Cao, S., Yang, B., & Pan, W. (2001). Co-DARFAD: The collaborative mechanical product design system. *Proceedings. of the 6th International Conference on Computer Supported Cooperative Work In Design*, 163-168.

Huang, G. Q. & Mak, K. L. (1999[a]). Web-based morphological charts for concept design in collaborative product development. *Journal of Intelligent Manufacturing*, *10*, 267-278.

Huang, G. Q. & Mak, K. L. (1999[b]). Design For Manufacture and Assembly on the Internet. *Computer in Industry*, *38*, 17-30.

Hummel, K. E.& Brooks, S. L. (1986). Symbolic representation of manufacturing features for an automated process planning system. In S. Lu & R. Komanduri (Eds.), *Knowledge-Based Expert Systems for Manufacluring,* ASME, 233 - 243.

IBM Dictionary of Computing (2008). IBM Terminology. Retrieved January 20, 2011 from http://www-01.ibm.com/ software/globalization/terminology/d.html.

International Standard Organization (ISO) (1992). Industrial Automation System and Integration – Product Data Representation and Exchange. Technical Report, Geneva, Switzerland: ISO Central Secretariat.

Jagannathan, V., Almasi, G., & Suvaiala, A. (1996). Collaborative infrastructures using the WWW and CORBA-based environments. *Proceedings of the IEEE Workshops on Enabling Technologies Infrastructure for Collaborative Enterprises* (*WET ICE'96*), 292-297.

Jagannathan, V., Dodhiawala, R., & Baum, L. S. (1989). Blackboard architectures and applications. San Diego, USA: Academic Press.

Li, W. D., Ong, S. K., & Nee, A. Y. C. (2006). *Integrated and collaborative product development environment: Technologies and implementation*. Singapore: World Scientific.

Lin, J., Fox, M. S., & Bilgic, T. (1996). *A requirement ontology for engineering design*. Unpublished technical report, University of Toronto, Canada.

Liua, C. L., Lua, Z. Z., Xub, Y. L., & Yuec, Z. F. (2005). Reliability analysis for Low cycle fatigue life of the aeronautical engine turbine disc structure under random environment. *Materials Science and Engineering*, *A*, 218-225.

Mahmood, Z. & Hill, R. (2011). *Cloud Computing for enterprise architectures*. London, England: Springer-Verlag.

McCarthy, D. R. & Dayal, U. (1989). The architecture of an active database management system. *Proceedings of ACM SIGMOD International Conference on Management of Data*, 215-224.

Ming, C., Jinfei, L., Qinghua, K., & Yuan, Y. (2010). Study on Collaborative Product Design Based on Modularity and Multi-agent. *Proceedings of the 2010 14th International Conference on Computer Supported Cooperative Work in Design*, 274-279.

National Institute of Standards and Technology (2012). The NIST definition of Cloud Computing. Retrieved February 3, 2011 from http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf.

Object Management Group, (2008). *Common Object Request Broker Architecture* (*CORBA*) Specification: Version 3.1. Needham, USA: OMG.

OpenNebula (2010). About the OpenNebula technology. Retrieved March 3, 2012 from http://www.opennebula.org/about:technology.

Roller, D., Bihler, M., & Eck, O. (1997). SN: Active, distributed knowledge base for Rapid Prototyping, *Proceedings of 30th ISATA - Volume "Rapid Prototyping in the Automotive Industries"*, 253-262.

Roller, D., Eck, O., Bihler, M., & Stolpmann, M. (1995). An Active Semantic Network for supporting Rapid Prototyping. *ISATA Proceedings on Mechatronics - Efficient Computer Support forEngineering, Manufactoring, Testing & Reliability, 41-48.*

Roller, D. & Eck, O. (1997). Constraint propagation using an Active Semantic Network. *Proceedings of 1st International Workshop on Geometric Constraint Solving & Applications.*

Roller, D. & Eck, O. (1998). Constraint propagation using an Active Semantic Network. In B. Brüderlin & D. Roller (Eds.), *Geometric Constraints Solving and Applications*, Berlin, Germany: Springer-Verlag.

Rollins, J. M., (2012). Speed of a piston. Retrieved March 15, 2012 from http://www.camotruck.net/rollins/piston.html.

IBM Dictionary of Computing (2008). IBM Terminology. Retrieved January 20, 2011 from http://www-01.ibm.com/software/globalization/terminology/d.html.

Schmidt, K. & Bannon, L. (1992). Taking CSCW seriously. *Computer Supported Cooperative Work*, *1*, 7-40.

Shah, J. J. & Mantyla, M. (1995). *Parametric and feature-based CAD/CAM*: *Concepts*, *techniques*, *and applications*. Toronto, Canada: John Wiley & Sons, Inc.

Shen, W. & Barthes, J. P. (1995). DIDE: A multi-agent environment for engineering design. *Proceedings of the 1st International Conference on Multi-Agent Systems* (*ICMAS'95*), 344-351.

Shen, W., Hao, Q., & Li, W. (2008). Computer supported collaborative design: Retrospective and perspective. *Computers in Industry*, *59*, 855–862.

Sprow, E. (1992). Chrysler's Concurrent Engineering challenge. *Manufacturing Engineering*, *108*, 35-42.

Thiel, S., Dalakakis, S., & Roller, D. (2005). A learning agent for knowledge extraction from an Active Semantic Network. *World Academy of Science, Engineering and Technology*, *7*.

Toye, G. , Cutkosky, M., Leifer, L., Tenenbaum, J., & Glicksman, J. (1993). SHARE: A methodology and environment for collaborative product development. *Proceedings of the IEEE 2nd Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 33-47.

Verroust, A., Schonek, F., & Roller, D. (1992). A rule oriented method for parametrized Computer Aided Design. *Computer Aided Design*, *24*, 531-540.

Wallis, A., Haag, Z., & Foley, R. (1998). A multi-agent framework for distributed collaborative design. *Proceedings of the IEEE Workshops on Enabling Technologies Infrastructure for Collaborative Enterprises* (*WET ICE'98*), 282-287.

Wang L. (2009). *Virtual environments for Grid Computing*. Karlsruhe, Germany: Universitaetsverlag Karlsruhe.

Wang, L., Shen, W., Xie, H., Neelamkavil, J., & Pardasani, A. (2000). Collaborative conceptual design: A state-of-the-art survey. *Proceedings of the 6th International Conference on Computer Supported Cooperative Work in Design*, 204-209.

Weiss, G. (1999). *Multiagent systems*: *A modern approach to distributed artificial intelligence*. Massachusetts, USA: MIT Press.

Wohlstadter, E. & Tai, S. (2009). *Web Services*, *Reference Entry*, *Encyclopedia of Database Systems* (*EDBS*). Heidelberg, Germany: Springer.

World Wide Web Consortium (2002). Web Services Architecture Requirements: W3C Working Draft 29 April 2002. Retrieved February 10, 2011 from http://www.w3.org/TR/2002/WD-wsa-reqs-20020429.

Xie, H. (2001). Tracking of design changes for collaborative product development. *Proceedings of the 6th International Conference on Computer Supported Cooperative Work in Design*, 175-180.

Xuan, W., Xue, S., & Ma., T. (2009). Architecture of collaborative design based on grid. *Proceedings of the Third International Conference on Genetic and Evolutionary Computing*, 244-247.

Zdrahal, Z. & Domingue, J. (1997). The world wide design lab: An environment for distributed collaborative design. *Proceedings of the 1997 International Conference on Engineering Design*, 19-21.

DEFINITIONS OF KEY TERMS

Active Semantic Network (ASN): A shared database system developed for supporting designers during product development and is realized as an active, object-oriented database system. ASN comprises a network of nodes and links, where the nodes represent objects of the real world and the links relations amongst these objects.

Adaptive: The ability of the intelligent agent to adapt with changes in its environment that commonly sources from changes of structure, program, or data (i.e. based on its input or in response to information) in such a manner that the expected future performance will improve.

Agent: An autonomous, reactive, pro-active computer system, typically with a central locus of control, that is at least able to communicate with other agents via some kind of communication language. Another common view of an agent is that of an active object or a bounded process with the ability to perceive, reason, and act. Various attributes are discussed in the context of agent-based systems.

Cloud Computing: By using virtualized computing and storage resources and modern Web technologies, cloud computing provides scalable, network-centric, abstracted information technology (IT) infrastructures, platforms, and applications as on-demand services. These services are billed on a usage basis.

Computer Supported Collaborative Design: The process of designing a product through collaboration among multidisciplinary product developers associated with the entire product life cycle.

Feature Taxonomy: A natural structure of feature-based manufacturing used to develop the feature library and to simplify and encourage the extension of the library.

Data Dictionary: A centralized repository of information about such data like meaning, relationships to other data, origin, usage, and format.

ISO/OSI Model: An an Open Systems Interconnection (OSI) model developed by ISO (International Organization for Standardization) in 1984, defines a networking framework for implementing protocols in seven layers. Control is passed from one layer to the next, starting at the application layer in one station, and proceeding to the bottom layer, over the channel to the next station and back up the hierarchy.

Multiagent System: A collection of software agents that communicates and works in conjunction with each other. They may cooperate or compete with others, or some combination of cooperation and competition, to solve a complex problem which is beyond the capability of each individual system.

Web Technology: The platforms and technologies which are used to develop a mechanism that allows two of more computer devices to communicate over a network through alternatives Web sites and Web applications.