Springer

Journal of Mechanical Science and Technology 22 (2008) 2180~2189

Journal of Mechanical Science and Technology

www.springerlink.com/content/1738-494x
DOI 10.1007/s12206-008-0716-y

Development of an authoring framework for the simplified customization of PDM systems †

In-Ho Song[1], Jeongsam Yang[2,*] and Beom Park[2]

[1]*Department of Mechanical Engineering Carnegie Mellon University 5000 Forbes Avenue, Pittsburgh, PA 15213, U.S.A*
[2]*Division of Industrial & Information Systems Engineering Ajou University San 5, Wonchun-dong, Yeongtong-gu, Suwon 443-749, Korea*

(Manuscript Received August 14, 2007; Revised July 15, 2008; Accepted July 17, 2008)

Abstract

Given that the amount of product data in firms is explosively increasing, a PDM system for effective data management is considered indispensable for product development. However, considerable time and specialized human resources are needed to customize a generic PDM system for satisfying the specific requirements of individual firms. To overcome this problem, we propose the use of UML object models in a PDM authoring framework. A PDM authoring framework, which provides authoring functions for the effective customization of PDM systems, will reduce the need for the intervention of PDM specialists in the design of the object models of the PDM system. We describe how a PDM authoring framework may be designed by using UML object models, and show how model-oriented application development (MOAD), in conjunction with the PDM authoring framework, can be used to build object models into a PDM system. Furthermore, we confirm the value of the framework by evaluating its performance under several conditions.

Keywords: Authoring framework; Object model; Product data management (PDM); Unified modeling language (UML)

1. Introduction

In light of the explosive increase in the amount of product data in firms, many companies have invested in product data management (PDM) systems to create a computing infrastructure that automates various phases of the product development process. The introduction of a PDM system for effective data management is critical for the product development process; however, the time and cost that are involved in the customization of PDM systems is a major stumbling block. When a PDM system is implemented, its functions should comply with the work processes of the company in question. Such compliance, however, involves a tedious and time-consuming coding of jobs because the PDM integration specialist is often rushed in to implement specific functions.

Commercial PDM systems use various approaches to overcome this problem. For instance, they create a database schema in a manner similar to that for object models in unified modeling language (UML), and they use a database engine to handle objects for searching and adding data. However, because these PDM systems use a nonstandard object model, they do not adequately satisfy the various complex requirements of a company that is attempting to implement a PDM system. In other words, the modeling of a company's information is severely restricted by a PDM system that cannot support multiple inheritances or various complex relations between objects. For these reasons, a discrepancy occurs between the information structure of a company and that of the PDM system that is implemented, which results in the diminished utility of the company's custom-built PDM system.

In the PDM literature, Thimm et al. [1] asserted

† This paper was recommended for publication in revised form by Associate Editor Dae-Eun Kim

*Corresponding author. Tel.: +82 31 219 2335, Fax.: +82 31 219 1610
E-mail address: jyang@ajou.ac.kr

that UML-based PDM modeling is suitable for modeling business processes. They claimed that this type of modeling, which is related to diverse methods of information-rich representation, can express models of various phases; they illustrated their method by modeling the design process of a vacuum cleaner. Sharma [2] used an IT-based, PDM framework to study collaborative product innovation. Huang et al. [3] used JAVA and J2EE technology to develop a Web-based system of PDM. Eynard et al. [4] implemented a UML-based PDM system and applied it to a product data structure and workflow. Oh et al. [5] suggested a mapping method, which was based on UML and STEP, for integrating heterogeneous CAD systems with PDM systems. Rezayat [6] investigated the implementation of an enterprise Web portal that was based on the Web standard and Web protocol, CORBA/DCOM, which is a distributed object standard, as well as JAVA, XML and IIOP. Sheng et al. [7] studied how PDM can be implemented on the basis of object-oriented programming technology. Sudarsan et al. [8] extended the core product model of the National Institute of Standards and Technology, while researching a method for modeling product lifecycles. Most of the related works either featured approaches for designing PDM schemas or specifications or dealt with applications for solving operational problems in the deployment of PDM systems. Further research is needed to directly resolve problems in the development of PDM systems; . in In particular, research on PDM authoring tools should facilitate the implementation of PDM systems for meeting the various complex requirements of individual companies.

We propose a PDM authoring framework that relies on a UML-based object model. Further, to enhance this authoring framework, we develop a PDM customization tool called model-oriented application development (MOAD). The MOAD authoring tool obviates the need for directly coding jobs, and thereby allows PDM engineers to concentrate on system modeling. In addition, because the MOAD authoring tool can be managed within a single system during the customization of a PDM system, the PDM deployment process is standardized and improves the efficiency of application development. In the on-site implementation of a PDM system, a consultant from the PDM firm creates or modifies the data schema by using our method and the MOAD authoring tool, while an engineer in the field can continue the remaining work after the modification of data.

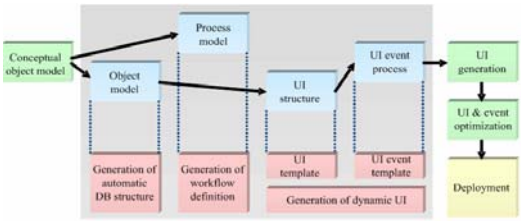

Fig. 1. PDM customization process of the MOAD authoring tool.

Fig. 1 shows the process of MOAD, which uses the PDM authoring framework to build object models. When a PDM conceptual model is concretized, during the PDM implementation process, as an object model through the MOAD authoring tool, the object model is generated inside the MOAD authoring tool through an automatic database structure generator. For each template, a dynamic user interface (UI) generator creates a UI structure and a UI event process. The process of PDM customization is finally completed by the creation and definition of events in relation to the UI and the optimization process. While a conventional PDM system performs this process through direct programming, our method relies on an object modeler by using the PDM authoring framework.

2. Design of a server in the PDM authoring framework

2.1 Configuration

The MOAD, in conjunction with a PDM authoring framework, can load various types of server modules, depending on the architecture of the PDM system. As shown in Fig. 2, a an MOAD server has the following seven modules that meet the functional requirements of a PDM system: a transaction controller; , a vault controller; an object engine; a dynamic database schema generator; a workflow engine; a name and directory service; and a workspace manager. Table 1 describes in detail the functional purpose of these modules. One MOAD server and one or more MOAD clients in a PDM system communicate with each other by using a network protocol called an Intelligent intelligent Information information Pipeline pipeline (IIP). The transaction controller manages the connection with the database through the dynamic database schema generator, and the vault controller is linked with a physical file server into which CAD data and documents are loaded.

Table 1. Key functions of the MOAD server.

Functions of the server	Description
Transaction controller	Manages the connection pool that connects the database and the distributed transaction
Vault controller	Functions as a sort of logical file server and is responsible for archiving physical files such as documents and CAD data
Object engine	Functions as a UML-based engine for handling objects and is responsible for managing class definitions and class relations; also provides a function for searching objects
Dynamic database schema generator	Performs the function of applying a logical structure that is defined as an object engine to a physical database table
Workflow engine	Functions as a workflow engine with a W*f*MC (Work work Flow flow Management management Coalitioncoalition) specification [11] and creates workflows
Name and directory service	Provides a directory service similar to a lightweight directory access protocol or active directory [12]
Workspace manager	Manages the personal work environment that is created for each MOAD user

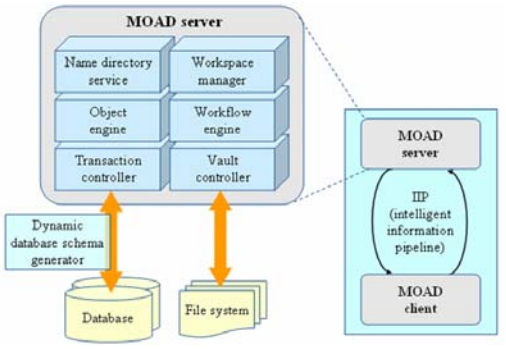

Fig. 2. Framework of the MOAD server.

2.2 UML-based object engine

The object engine in the MOAD authoring tool enables us to define object models that can effectively express the relations between object models. In addition, because this object engine can express UML entities, such as multiple inheritance and association among objects, it can reflect the unique data structure of a company to the company's own PDM system. As shown in Fig. 3, the structure of the object engine consists of an object-definition repository for managing meta objects and an object-instance processor for managing object instances.

Table 2. The six functions of the object definition repository.

Functions	Description
Class definition	Defines objects and assigns their properties and operations
Relation definition	Defines relations between objects, such as associations or aggregations in the UML
Code definition	Provides functions for managing code-typed data
UI definition	Defines a method for expressing defined objects in the windows of a MOAD server and MOAD clients
Numbering rule	Defines rules for automatically creating a unique number for each object that is created in an object modeler
Event definition	Provides events for performing additional program codes at a specific point of time in the process of creating and modifying objects

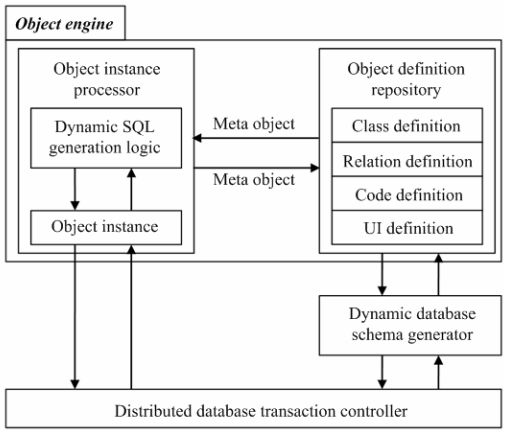

Fig. 3. Architecture of the object engine in the MOAD authoring tool.

2.3 Object definition repository

The object-definition repository, which defines, saves and manages objects, is designed on the basis of a UML class diagram and includes all of the UML concepts. As shown in Table 2, the object-definition repository manages information under the following six classifications: the class definition; , the relation definition; the, code definition; the, UI definition; the, numbering rule; , and the event definition. The physical database schema and the object-definition repository are synchronized on the basis of information that is saved in the latter. In addition, when the object engine begins to run, all the information that is managed by the object-definition repository is copied from the database to memory. The checking and searching of an object in the MOAD authoring tool

can be done without changing the information in the repository. Thus, a rapid response is possible for all memory processing.

2.4 Object-instance processor

The object-instance processor, which is based on the properties that are saved in the object-definition repository, creates and manages the data in each object. As shown on the right side of Fig. 3, the application program interfaces (APIs) of the object-instance processor, which are used for creating objects or searching for a particular object in the object-definition repository, dynamically create a suitable SQL (Structured structured Query query Language-language) statement by using information from a MOAD client.

The object-instance processor has a built-in object-instance cache for enhancing the access performance of an object engine. When object data are first accessed, they are saved in a cache. The data are maintained in the cache until any changes are made to the data or until the cache space is full. Whenever any of the data in the cache is needed, it can be acquired from the cache with a considerably low processing load and without running a program for creating a complicated SQL statement. Thus, the object engine can improve the speed of response and processing capability.

2.5 Distributed database transaction controller

The distributed database transaction controller, which is assigned to several databases, performs distributed transactions in the MOAD server. As shown in Fig. 4, the object engine in the MOAD server uses the distributed transaction management function to manage the division of objects into several databases.

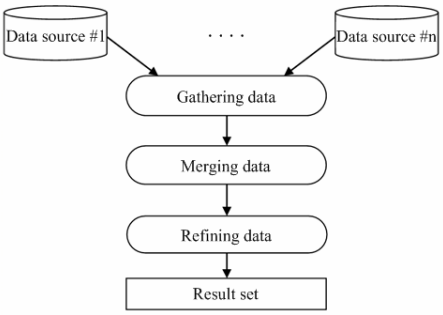

Fig. 4. Processing of distributed data sources for a distributed transaction.

Using the distributed management function of the distributed database transaction controller, the MOAD server assigns a data source that is to be saved in the database according to its class. The data source of each class is then used to generate SQL commands, such as those for creating and searching data. Furthermore, depending on the relations between classes, the server performs transactions by simultaneously accessing more than one data source. In this case, the object engine collects data by connecting individual data sources and by reprocessing the data that is stored in memory.

2.6 Object modeler

Fig. 5 shows a PDM system that was implemented by using the MOAD authoring tool. The MOAD can build various functions of the PDM system in accordance with customer-defined specifications of the following seven elements: the model; , the package; , the class; the field; , the action; , the field group; , and the reference.

Model The model, which is a root-element of six child-elements, has a unique identification (*Ouid* in Fig. 5) and foundation package. Based on the concept of an object model, the MOAD authoring tool expresses all the object models in the window as a tree-view list, and provides a framework for the customization of a PDM system by clarifying the structure and behavior of the PDM system. As shown in Fig. 5, whenever the object modeler is executed, the name of the model on the left side of the tree-view is visible, and each model has two tab menus, namely,: a model menu and an event menu.

Package A package, which consists of several class groups, is a common mechanism for grouping elements.

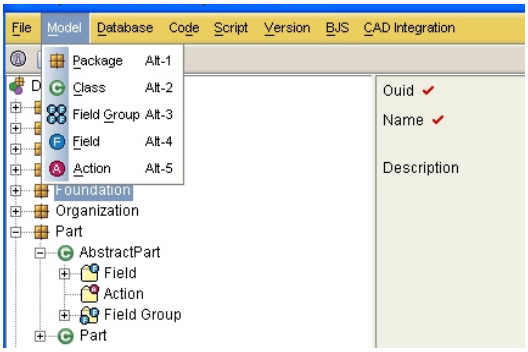

Fig. 5. Elements of the object modeler.

Class A class is a group of objects that share the same attributes, operations, relations and functional meaning. In the MOAD authoring tool, a user, who executes a MOAD client, creates individual objects that define various types of information, such as part information, CAD information, and document information. A class has individual items of information regarding relations, events and numbering rules.

Field A field is an abstract concept about the status of various kinds of data pertaining to object classes; the attributes of physical data in a database are saved in fields.

Action An action is an attribute for adding individual functions to facilitate user-customization. Unlike events, which are performed in specific situations that are assigned beforehand, an action can execute work by means of event-input through buttons that have been directly added to a client's UI.

Field group A field group has field items and action items. It is responsible for setting up the contents that are presented in tab items of the MOAD client's UI and for assigning the position of each tab item. A field group is inherited by a parent class, and its name can be overridden with that of the parent class.

Reference A reference defines the relation between two classes. The class relation can be described as an association, an aggregation or a composition. An association, which defines the name of the interrelated classes, represents the horizontal relation between two objects; additionally, if an object must have another relation, the association specifies the characteristics of that relation. In addition, the association is used to attach a part to a document in the MOAD. Fig. 6 illustrates an authoring process for defining an association in the MOAD. Firstly, after selecting the tab, *Relation* [annotation (a) in Fig. 6], the user writes down an instance-name for the association in the edit box [annotation (b) in Fig. 6], which is activated in the window that is at the bottom-right. Secondly, the user selects *Association* from the drop-down list of a combo box named *Type* [annotation (c) in Fig. 6] and inputs an identifier of the object into the edit-box, *Code* [annotation (d) in Fig. 6]. Lastly, the user writes down names that are represented in the object, *End1*, for a part [annotation (e) in Fig. 6] and in the object, *End2*, for a document.

An aggregation, which is similar to an association and which represents objects that are combined with other objects, is used to represent relations between the whole and the parts of several associations. To

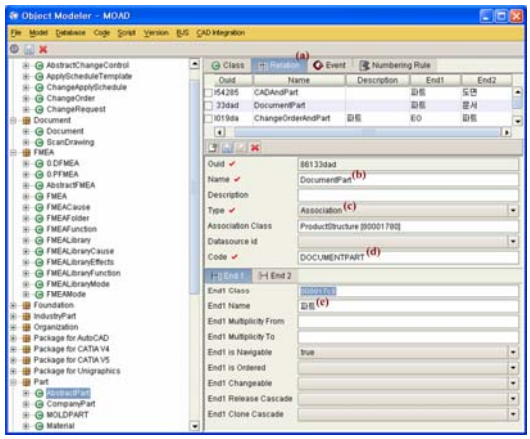

Fig. 6. Representation of the association relation between a part and a document by using the object modeler.

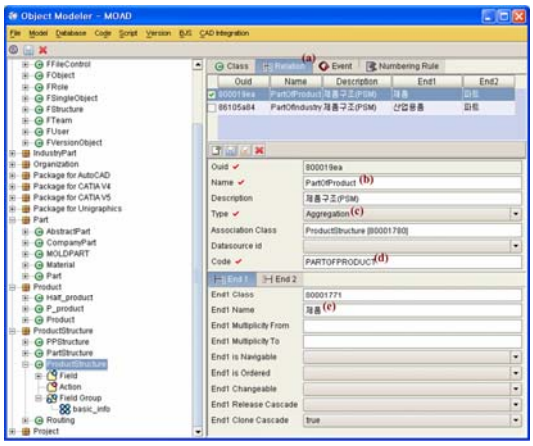

Fig. 7. Representation of the aggregation relation between a product and a part.

define an aggregation using the MOAD, firstly, the user first selects the tab, *Relation* [annotation (a) in Fig. 7], and writes down the relation name on the edit box [annotation (b) in Fig. 7]. Secondly, the user selects *Aggregation* among the elements in a drop-down list [annotation (c) in Fig. 7] and inputs an identifier of the object into an edit-box named *Code* [annotation (d) in Fig. 7]. Lastly, the user writes down names that are represented in the object, *End1*, for a whole product [annotation (e) in Fig. 7] and in the object, *End2,* for an individual part.

A composition is a type of aggregation. The aggregation and the composition, which represent a relation that associates an object with other objects, are used for the relation between a user and a department or between a work-breakdown structure and work-

scheduling. In the case of a simple aggregation, where a part can be shared by the whole, a composition represents cases in which a part has a strong relation with the whole [10].

3. Design of a client in the PDM authoring framework

3.1 Configuration

Fig. 8 shows the client structure of the MOAD authoring system. The structure consists of a dynamic UI generator, a script invoker, a workflow monitor, and an interface-adapter controller. Through an interface-adapter controller and an adapter for CAD integration, the MOAD client is linked to eight commercial CAD applications, namely,: CATIA V4, CATIA V5, ProEngineer, Unigraphics, Solidworks, Inventor, SolidEdge and I-DEAS. We used the APIs of the eight CAD systems to develop the interface-adapter controller and the adapter for CAD integration. Table 3 summarizes these modules.

Table 3. Key functions of the MOAD client.

Modules	Description
Dynamic UI generator	Relies on object models that are defined by an object engine to provide a visualization function for real-time composure of a client window
Script invoker	Executes client script files that are defined by an object engine and a workflow engine in real-time
Workflow monitor	Monitors processes that are being executed by a workflow engine in real-time
Interface adapter controller	Loads adapters for interfacing with external programs such as CAD applications and clients, and controls loaded adapters
Adapter for CAD integration	Integrates the interface with commercial CAD applications and clients

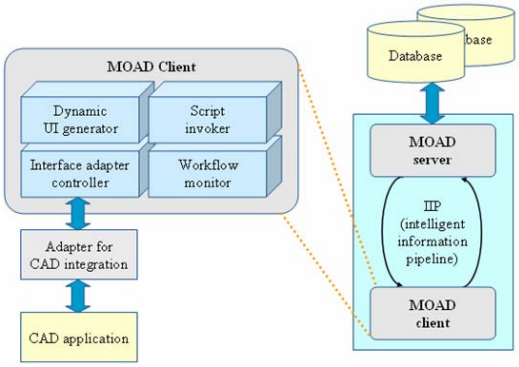

Fig. 8. Architecture of the MOAD client.

3.2 Dynamic UI generation

One of the critical factors of in customizing a PDM system is to establish a user-friendly interface that enables users to effectively access the PDM system. Although UIs have the important capability of making a data structure, they are often overlooked during the customization of a PDM system. Hence, to make all UIs within the PDM system operate in a normalized sequence and to enable users to enhance the usability of the data, we need to determine the optimal UI configuration. Notwithstanding rich-client applications and the Web-based environment, dynamic UI generation should be provided for users with the same view.

To apply this requirement to a an MOAD client, we propose the use of a window-composition tool called a UI builder, which can effectively provide relevant information to users. We developed the UI builder after analyzing the windows and the UI structure of commercial PDM systems. As shown in Fig. 9, the UI builder uses an object engine to save all the information about window composition on a an MOAD server. The UI builder gathers the information about window composition from an object-handling engine and automatically creates a user-required UI in real-time. In addition, the UI builder dynamically creates a an HTML page that operates in the same way as a page in a Web environment. Thus, if a PDM administrator or developer modifies the information on window composition, the change of information is automatically applied to the rich-client application and the Web-based environment. Fig. 10 displays a search window for CAD data in the Web-based PDM system that is created through the UI builder for dynamic UI generation.

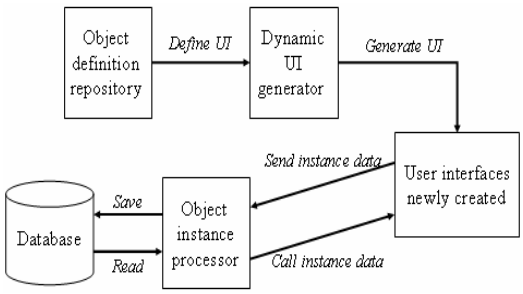

Fig. 9. Dynamic UI building processes of the UI builder.

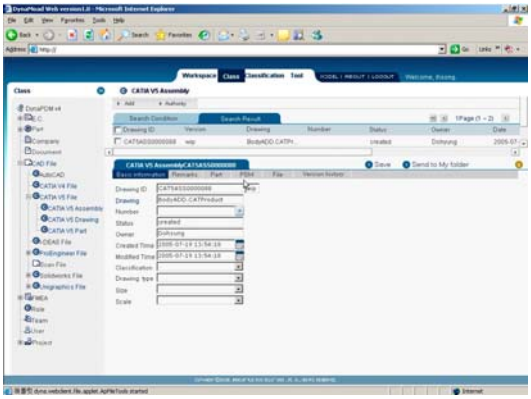

Fig. 10. A search window of the Web-based PDM system that is customized by the UI builder.

3.3 Intelligent information pipeline

In general, the greater is the amount of data that is stored in a database, the slower is the overall search speed. We introduce an IIP to the MOAD authoring framework, which does not affect total system performance even when the amount of data is progressively increasing. As shown in Fig. 8, an IIP is a network protocol that acts as a sort of remote procedure call at the lowest level of the MOAD authoring tool. After recognizing the type of data that is sent to a network, the IIP uses an intelligent transmission algorithm that selects the optimal transmission route for that type of data.

We introduce an object-handling engine for optimal performance and the creation of a database schema that is based on an object model. An object-handling engine uses database attributes to estimate the structure of the database schema and creates a suitable SQL statement for each request. In addition, if only one of various property values is changed, the object-handling engine checks and modifies the property to create an SQL statement that alters only the content of the modified property. Furthermore, the search for multiple items of data is more complicated than the search for a single item of data because multiple items of data might involve more than one class and might have a complicated inheritance relation. In this case, the object-handling engine analyzes the relations between all classes to identify the target class that is needed in the search. Finally, similar to the method used by an SQL optimizer in a database system, the object-handling engine creates an optimal SQL statement for searching. Due to the characteristics of the object-handling engine, our IIP can overcome the problem of slow response times in the face of increasing amounts of data.

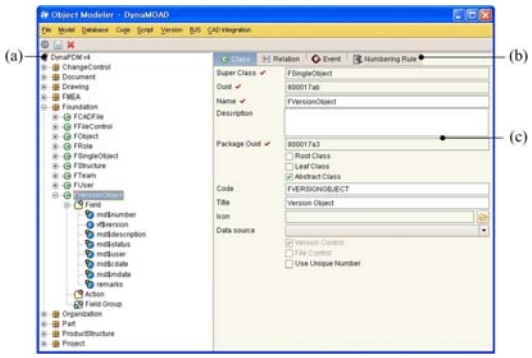

Fig. 11. Object modeler of MOAD.

4. PDM implementation with MOAD

Fig. 11 shows an object modeler in MOAD under the PDM authoring framework. Annotation (a) of Fig. 11 indicates a hierarchical, tree-view list of objects and information, such as objects and fields that are derived from the authoring framework. The node, *DynaPDM*, refers to a root node that is implemented with the model in the object modeler, as described in section 2.6. The other nodes, *ChangeControl*, *Document* and *Drawing*, were implemented by the package in the object modeler. The child-nodes refer to nodes of the *Foundation* package such as *FCADFile*, *Ffile-Control* and so on, which are created by the class function in the object modeler. Finally, the node, *FVersionObject*, which includes the number, version, description of the representative type, and the status of each datum, is created by the field function in the object modeler. Annotation (b) of Fig. 11 shows a menu of the defining relations between classes ; this menuthat can be used to define references, events and the numbering rules. Annotation (c) of Fig. 11 shows an input window for the attributes of each object.

Fig. 12 shows a workflow modeler of MOAD. Annotation (a) shows a document-approval business process that is generated by the workflow modeler of MOAD, annotation (b) indicates the activities that constitute a sublevel of a business process, and annotation (c) shows a graphical process-editor for creating or editing a business workflow in a graphic environment. By using these three functions, we can implement a business process that complies with the various types of requirements of a company.

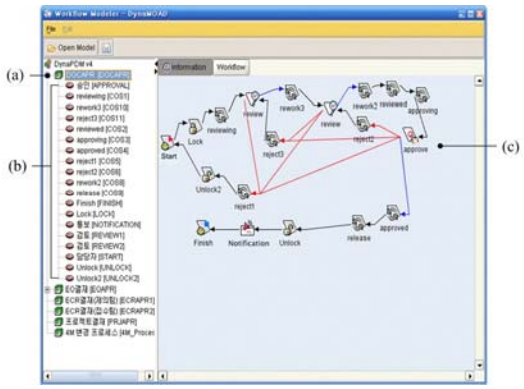

Fig. 12. Workflow modeler of MOAD.

5. Evaluation of the search performance

To evaluate the performance of the MOAD authoring tool, we assigned the following target objects:

(1) The creation of one abstract class, as shown in Fig. 13, followed by the creation of 15 leaf classes that are inherited from the abstract class.

(2) The selection of all the leaf classes from the same data source whenever the leaf classes are created.

(3) The creation of the same property and a different property, respectively, for each leaf class.

Table 4 shows the environment of the client system and the server system that we used to evaluate the performance. The four steps of the evaluation were as follows. First, after searching all the class data in a database of 60,000 items of class data, we measured the time that elapsed before the search results were displayed on a client window. As shown by annotation A (a) of Fig. 14, the elapsed time was 10.5 seconds. Second, for 40,000 items of class data stored in the database, the search lasted 6.3 seconds before all the data was displayed on a client window (as shown by annotation B (b) of Fig. 14). Third, 10,000 results were displayed on a client window when we specified a search condition for 60,000 items of data in the database. In this case, as shown by annotation C (c) of Fig. 14, the search lasted 1.7 seconds. Fourth, 1,000 results were displayed on a client window when we specified a search condition for a database of 60,000 items of class data, and the elapsed time for displaying the search results was less than 1 second, as shown by annotation D (d) of Fig. 14.

To achieve a reasonable evaluation, we repeated the test six times and after discarding the best and worst results, we averaged the remaining measurements. The evaluation results confirm that the search and viewing speed of our authoring system is between eight and 17 percent faster than that of a commercial PDM system under the same conditions.

Table 4. The system environment that is used for the evaluation of the search performance.

Database server	Windows XP Pro, Intel Pentium 4 2.8 GHz, 1 GB RAM Oracle 9i R2 (9.2.0.1.0)
Application server	Windows XP Pro, Intel Pentium 4 2.8 GHz, 1 GB RAM MOAD Server v1.0 Java™ 2 SE 1.4.2_05 HotSpot™ Server VM
Client	Windows XP Pro, Intel Pentium 4 2.8 GHz, 1 GB RAM MOAD Client v1.0 Java™ 2 SE 1.4.2_05 HotSpot™ Client VM
Network	100 Mbps Ethernet

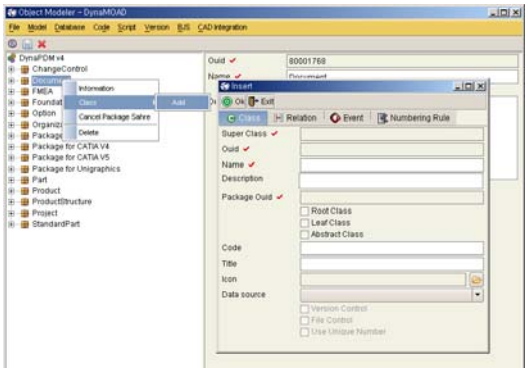

Fig. 13. The creation of an abstract class with the MOAD authoring tool.

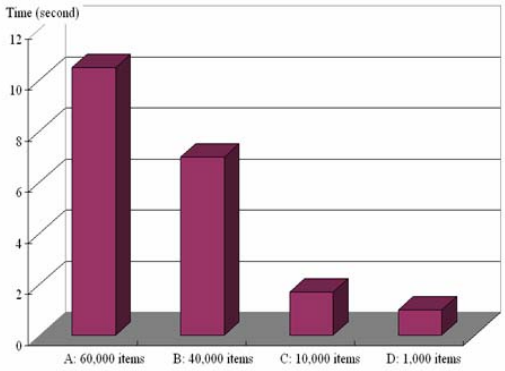

Fig. 14. Evaluation results of the search performance.

6. Conclusion

In recent times, several companies have invested in PDM systems for the purpose of conducting and managing various phases of the product development process as a unified process. However, because a PDM system should be implemented in compliance with each company's own work processes, the implementation is time-consuming and requires many PDM specialists. For this reason, the introduction and operation of the PDM system in a company tends to cause a bottleneck. To overcome these problems, we proposed a PDM authoring framework that enables us to use UML object models to simplify the customization of PDM systems.

In sum, we adopted the following, three-pronged approach to implement the PDM authoring framework.

(1) We developed a UML-based, object modeler for the design of the PDM authoring framework. Our object modeler and the MOAD authoring tool enable the data structure of a company to be applied to the PDM system, particularly with respect to the relations between objects, such as the multiple inheritance that results from UML modeling.

(2) We proposed a UI builder for the convenience of UI manipulation in the client application of the PDM system. Through the UI builder, we can automatically create a suitable UI in real-time once the object engine provides the requisite information on the window layout of the PDM system. This capability enables users, who lack programming skills, to participate in the development of the PDM system; it can also reduce the time and cost that is involved in customizing the PDM system.

(3) We applied a sort of communication protocol called an IIP to the MOAD authoring tool. Our evaluation of the search performance confirms that the IIP can expedite searches, even for large databases.

The MOAD authoring tool has already been commercialized and research is currently underway on a script-based, PDM platform to enable more effective authoring functions.

(4) We applied a sort of communication protocol called an IIP to the MOAD authoring tool. Our evaluation of the search performance confirms that the IIP can expedite searches, even for huge databases.

The MOAD authoring tool has already been commercialized and research is currently underway on a script-based PDM platform for more effective authoring functions.

Acknowledgement

This research is supported by the Foundation of ubiquitous Ubiquitous computing Computing and networking Networking project (UCN) Project, the Ministry of Knowledge Economy (MKE) 21st Century Frontier R&D Program in Korea and a result of subproject UCN 08B3-O3-20S.

References

[1] G. Thimm, S. G. Lee and Y. S. Ma, Towards unified modeling of product life-cycles, *Computers in Industry,* 57 (2006) 331-341.

[2] A. Sharma, Collaborative product innovation: integrating elements of CPI via PLM framework, *Computers-Aided Design,* 37 (2005) 1425-1434.

[3] M. Y. Huang, Y. Y. Lin and H. Xu, A framework for Web-based product data management using J2EE, *International Journal of Advanced Manufacturing Technology,* 24 (2004) 847-852.

[4] B. Eynard, T. Gallet, P. Nowak and L. Roucoules, UML based specifications of PDM product structure and workflow, *Computer in Industry,* 55 (2004) 301-316.

[5] Y. C. Oh, S. H. Han and H. W. Suh, Mapping product structures between CAD and PDM systems using UML, *Computers-Aided Design,* 33 (7) (2001) 521-529.

[6] M. Rezayat, The enterprise-Web portal for life-cycle support, *Computers-Aided Design,* 32 (2) (2000) 85-96.

[7] B. Sheng, J. Yu, Z. Ma and Z. Zhou, Research on the modeling of PDM system based on object-oriented technique, In: *The 8th International Conference on Computer Supported Cooperative Work in Design, 2004*, 747-752, Xiamen, China (2004).

[8] R. Sudarsan, S. J. Fenves, R. D. Sriram and F. Wang, A product information modeling framework for product lifecycle management, *Computer-Aided Design,* 37 (2005) 1399-1411.

[9] I. -H. Song and S. -C. Chung, Geometric kernel design of the Web-viewer for the PDM based assembly DMU, *Transaction of KSME (A),* 31 (2) (2007) 260-268.

[10] Object Management Group (1997). The Unified Modeling Language Release 1.1, Reference Manual. Available at http://www.omg.org/.

[11] The Workflow Management Coalition. Available at http://www.wfmc.org.

[12] Lightweight Directory Access Protocol RFC 2251. Available at http://www.faqs.org/rfcs/rfc2251.html.

Inho Song is a postdoctoral associate in the Department of Mechanical Engineering, Carnegie Mellon University, USA. He received the Ph.D. degree in Mechanical Engineering from Hanyang University, Seoul, Korea in 2007. From 2002 to 2007, he served as a CAx team leader of the INOPS Company (CIES R&D Center), Seoul, Korea. He has developed the sketch-based CAD system for an automotive company. His research interests include collaborative design, sketch-based CAD, geometry translation, geometry compression, product data exchange, PDM/PLM, digital manufacturing, and virtual reality.

Jeongsam Yang is an assistant professor in the Department of Industrial & Information Systems Engineering and is leading the CAD laboratory(http://cad-lab. kaist.ac.kr) at Ajou University. He worked at Clausthal University of Technology (Germany) as a visiting researcher and the University of Wisconsin-Madison (USA) as a postdoctoral associate. He obtained his Ph.D. in mechanical engineering in 2004 at KAIST. His current research interests are product data quality (PDQ), VR application in product design, product data management (PDM), knowledge-based design system, and STEP.

Peom Park is a professor in the Department of Industrial & Information Systems Engineering and is leading the Human Technology Research Center and Human Factors/HCI laboratory (http://hci.ajou.ac.kr) at Ajou University. He worked on HCI and Telecommunication system at ETRI as a senior researcher. He obtained his Ph.D. in Industrial and Manufacturing Systems Engineering at Iowa State University on 1992. His current research interests are uT applications, u-Healthcare/Telemedicine, Telematics and Ergonomic/Safety Design.