P2M2 – A Project Memory Management System

Hatem BEN STA

LI3 – Laboratoire de l'Ingénierie Intelligente des Informations,
Ecole Nationale des Sciences de l'Informatique
Université la Manouba
Campus Universitaire de la Manouba, la Manouba 2010

Abstract
The knowledge preservation of design project is an important issue for high technological industry. A general framework for managing knowledge pertaining to design projects is proposed. The objective is to allow preserving relevant information to be used later, thanks to a product design project memory. We introduce a generic platform-independent model that fits requirements of project memory management; this platform is based on UML generic models. We describe P2M2, an original implementation over object-relational technology in a multi-tier architecture; model transformation based on Model Driven Architecture (MDA) is developed. Finally, we discuss the benefits of ontology engineering for Project Memory management purposes.
Keywords**:** *Project Memory, Object-Relational, Knowledge Management, Ontology.*

1. Introduction

In design projects characterized by a high level of technological and organizational complexity, an important question is to know how a company or partners consortium can maintain competencies generated during an engineering project until their next use, and how the technical choices validated or invalidated within a previous projects can be reused.
In the framework of high technology products, designers use their experience to make choices. The justification of these choices is sometimes stored in documents which are exploitable with difficulty or at the very worst remains in the implicit domain and is never formalized. These choices depend on the context: economic constraints for example can change during the time; new technologies can appear, as well as new requirements concerning comfort or security.
If a team has to design a new product version some time later, it is interesting to retrieve the different considered solutions and the reason why they were rejected. The cost of a particular component was perhaps prohibitive, but it is now acceptable, and so a previously abandoned solution becomes possible.

So, the traceability of reasoning process can ensure going faster in the design of future products, by reusing. Furthermore, it is often necessary to know who by, when and why, the decision was made, to determine the responsibilities when a problem occurs.

Another problematic is to preserve information about a product configuration. An aircraft for example is often a particular case, and to be able to ensure the airplane's maintenance, a company has sometimes to call actors who have left the firm, because these persons are the only ones to have the appropriate knowledge. This situation is unacceptable.

So, the question is: how to store design choices, their justifications, working hypotheses, context and authors of these choices. The proposed answer is to implement a method of expert's knowledge management using a software tool: a product design project memory [2]. This software must be considered as a means serving a learning organization, including human aspects. In this paper only technical aspects are considered.

2. A Model for Project Memory

2.1 Project Memory

A project memory can be defined as a memory of both knowledge and information acquired during

the realization of projects [5]. [13] distinguishes the memory of the projects characteristics (concerning the context, the organization, the result) from the memory of the design reasoning (relating to the decisions and the resolution of problem). In the field of the engineering and knowledge management, some traditional methods can be quoted, like REX [12] (individual memory of experiment), MKSM [6] and CommonKADS [18] (memory of activities). Several approaches are proposed more specifically for the design memory. For example, "Design Rationale Capture Design" [11] based on a representation language (language DRCS) is adapted to concurrent engineering. Other approaches like IBIS [4] or QOC [3] are interested in the decision-making process in design.
Other non-technical aspects must be taken into account, as [10] shows that it is necessary to develop a new culture to transform an organization by project into a learning organization. [9] shows the interest to capitalize on the failures and missed opportunities. Many aspects require nevertheless to be improved, in particular the problem of the traceability of the design choices is still of topicality.

2.2 General Architecture of the Information System

Our approach consists in proposing a generic model allowing the implemen-tation of project memories. In reference to model levels proposed by [16], we suggest a three level architecture (fig. 1).

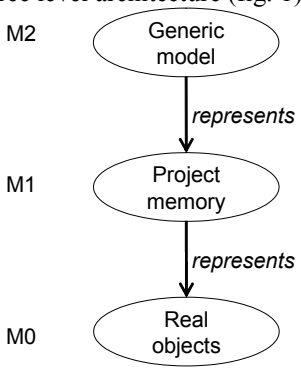

Fig. 1 The three levels architecture.

Level M0 is relative to real world "objects", such as product, human or material resources, calculation resources, CAD models, documentation, etc.
Level M1 corresponds to the models of the previous real objects, and constitutes a project memory; it structures the pertinent information without redundancies, but indicating where to find this information.

Lastly, level M2 is the more abstract one; it describes the model allowing project memories instantiation. This level is generic and ensures the possibility to memorize all projects' information. For example, it is possible to add dynamically attributes to a class, to associate it to one or several viewpoints. It also allows linking to the actor who is responsible of its values' validity and to elements from which it is issued (documentation, calculation resource, etc.). By instantiation of the model, the software's developer of a particular project memory for an enterprise has to define the appropriate information to store, depending on its next use. For example, if the concern is tracking the responsibilities in case of default, the software will store the tasks, the actors, the organization and the date of each action.

2.3 A generic data model

In the framework of product design project, real objects can often be represented by tree structures The search of a generic concept leads to represent such models by the use of a pattern (design scheme) fig. 2 adapted from those proposed in [7]. A pattern is in a sense a motif duplicable and adaptable according to necessities.

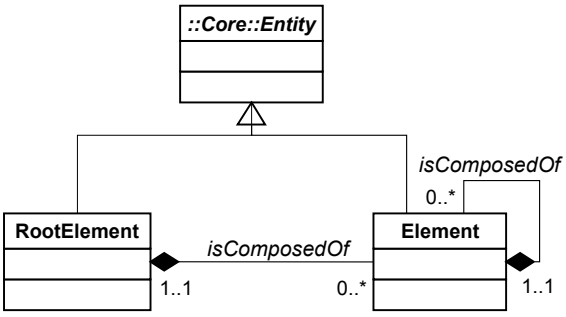

Fig. 2 Patterns describing a tree-like structure.

This pattern contains a class *Root-Element* intended to represent the root of the structure, because it was noticed that this one is stable. Indeed, if one represents the WBS of a project for example, the root is at once the name of the project and remains unchanged.
On the other hand, for Gamma, leaves have to remain leaves, without possibility of continuing the breakdown. This limit does not correspond to what was noticed in real design cases. The users often have to break down elements that were previously leaves. As a result, the pattern we propose contains a class called *Element* representing a node of the breakdown. This class contains a reflexive association (a node can consist of nodes).

The association between *Root-Element* and *Element* as well as the reflexive association represented on fig. 2 are of composition type; a node belongs only to a father node, and the suppression of this father

leads to suppress all children nodes. But the pattern can be adapted according to necessities, and one or some of these compositions can become aggregations; in this case, a node can be shared between several fathers, and the lifecycles of a node and its father can be different.

We may also notice that such a pattern can be enriched by constraints described with a language such as [15] to specify for example the impossibility to create a circular breakdown structure (if Element_1 consists of Element_2, then Element_2 can not consist of Element_1). These constraints are specified once.

The pattern described allows only keeping the last version of the arborescence, administering neither its history nor any other constraint such as exclusivity between children nodes, etc. These aspects are managed by the means of relations which are defined between entities in the package *Core* [2].

Fig. 3 gives the global view of the information system models structure in packages. This paper concerns only the package *Product*.

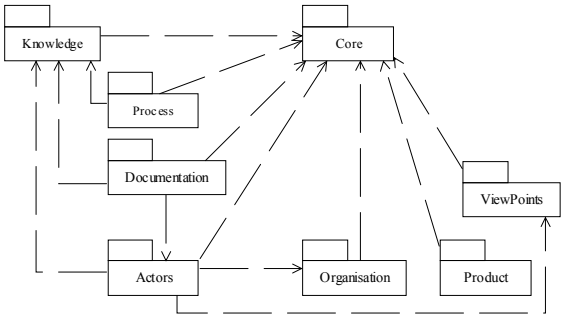

Fig. 3 Package structure of the IS models.

3. P2M2: An Object-Oriented Project Memory Management System

3.1 Prototype architecture

The models proposed are currently under implementation by using multi-tier J2EE architecture (fig. 4).
The first tier is the database tier supported by an Oracle database management system in version 9iR2 and is in charge of:

- The definition of the database type hierarchy using the object/relational feature of the server (type inheritance, nested tables, type references…)
- Data storage in one object relational table named "entity_tab"
- Data filtering with object views associated to database triggers "instead of…"

The second tier consists of a business application server hosting one entity EJB per object view and one session EJB implementing the classical façade pattern. This pattern provides a unified interface to a set of interfaces in a subsystem. Façade defines a higher-level interface that makes the subsystem easier to user [7].
The third tier is the presentation tier composed of JSP and html pages.

The last tier is the client tier. The client uses the application trough a simple web browser.

The fig. 4 illustrates the component mapping for the product aspect of the project memory.

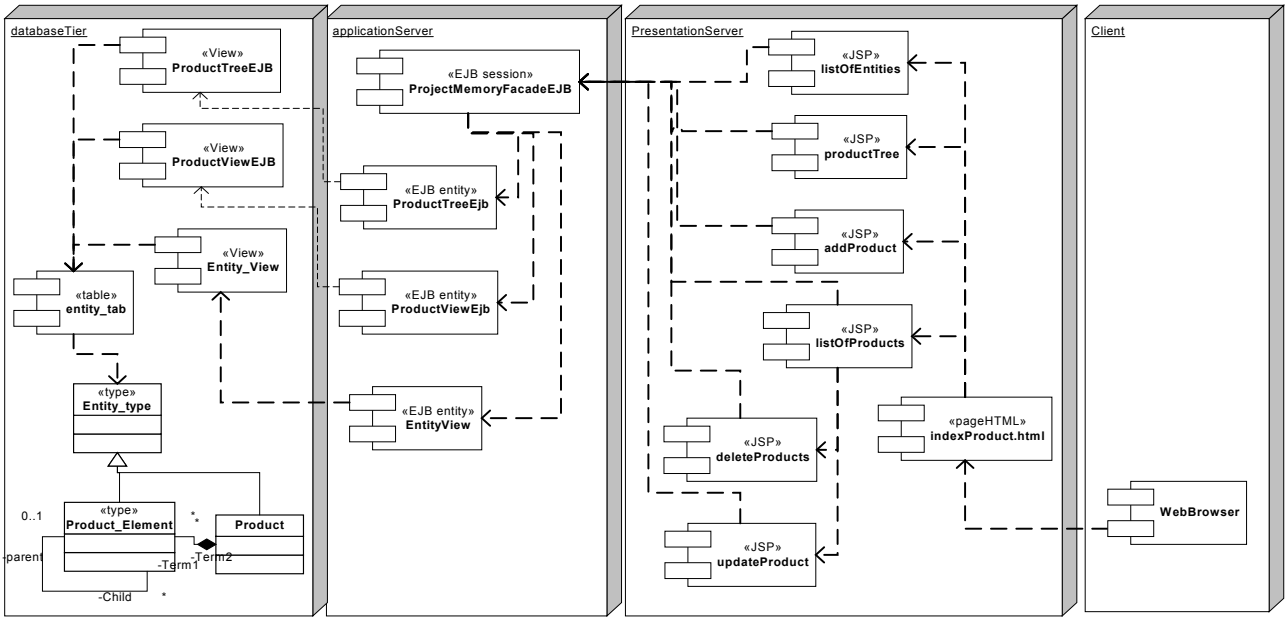

Fig. 4 General technical architecture of the information system.

For the implementation of the project memory, one of the main interesting features of this database management system is to provide the developer with the object relational technology. The object-relational model is based on the extension of the relational model by the essential concepts of the object. The system main part thus remains relational, but all the key concepts of the object are added there in a form particularly designed to facilitate the integration of the two models. In addition to built-in data types, the developer can define new object types that make it possible to model complex structures such as type hierarchy, object references, etc. In general, the object-type model is similar to the class mechanism found in C++ and Java. Like classes, objects make it easier to model complex, real-world business entities and logic, and the reusability of objects makes it possible to develop database applications faster and more efficiently. By natively supporting object types in the database, Oracle enables application developers to directly access the data structures used by their applications. Object abstraction and the encapsulation of object behaviours also make applications easier to understand and maintain [17].

3.2 Model Implementation Guidelines

The classical principle of the model implementation is the following:

1. For each UML class, define one user defined type ("**CREATE TYPE** MyType…" statement). For sub-classes, sub-types are created using the " **CREATE TYPE** … **UNDER** …" statement.

E.g, the statements for the diagram shown on fig. 5 are:

CREATE TYPE Class1Type(

Id_class1 **INTEGER**, Attrib1 **VARCHAR2(10)) NOT FINAL**

CREATE TYPE Class2Type

UNDER Class1Type (

Attrib2 **VARCHAR2(10))**

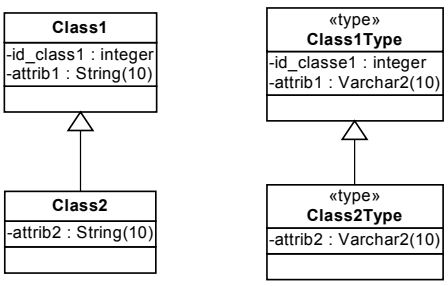

Fig. 5 Type structure for a class hierarchy.

2. Define one object-relational table per user-defined type ("**CREATE TABLE** MyTable **OF** MyType" statement), except for types corresponding to UML sub-classes.

- For each association (including aggregation and composition) and according to the multiplicities expressed on association-ends:
  1. Define references (**REF** or **FOREIGN KEY**)
  2. Define composite structure if necessary for managing "**1..***" multiplicities.
- For each inheritance relation it is possible to use two different implementation models: horizontal or flattened model.
  1. In the horizontal model, the developer defines one table per type (fig. 6)

Fig. 6 Horizontal model: one table per type.

The benefit of this model resides in the straightforward access to data. However it becomes difficult to build referential integrity constraints concerning instances of different classes of the hierarchy due to the scattering of the data in the large number of tables.

In the flattened model, the substitutability principle in a type hierarchy is used. This principle indicates that one instance of the subclass can be considered as one instance of its super-class, i.e. from set viewpoints all elements of the subclass also belong to the set of elements of the super-class. In practice, only one table of the super-class type can be created for storing data of all the type hierarchy (fig. 7). The statements of the data manipulation language can access directly to the columns corresponding to the attributes of the super-type, but have to include down casting operators (i.e. **TREAT**) to access to columns corresponding to the attributes of the sub-type.

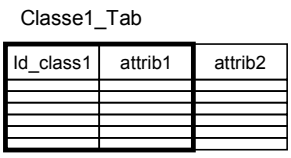

Fig. 7 Flattened model: one table for the hierarchy.

This last possibility was chosen for the inheritance implementation because all the project memory components are sub-classes of a unique root class

("Entity") and the management of links between two components of any class is easier this way.

3.3 Implementation

Several models are implemented in the prototype: product, documentation, evolution relation and version models. In this prototype it is possible to define (create, update, delete) different product lists (fig. 8 a) and hierarchies (fig. 8 b).

Also, it is possible to manage documentation, by using nested documentation items whose structure can be modified throughout versions (figure 9a). P2M2 allows one to reference as many entities as necessary in a version item (figure 9b), which is a composition of a set of any entity of the project memory. Versions can evolve: their history is being referenced in an evolution item. On figure 10, one can see how a version or a documentation item can be declared as the evolution of a former entity of the same type.

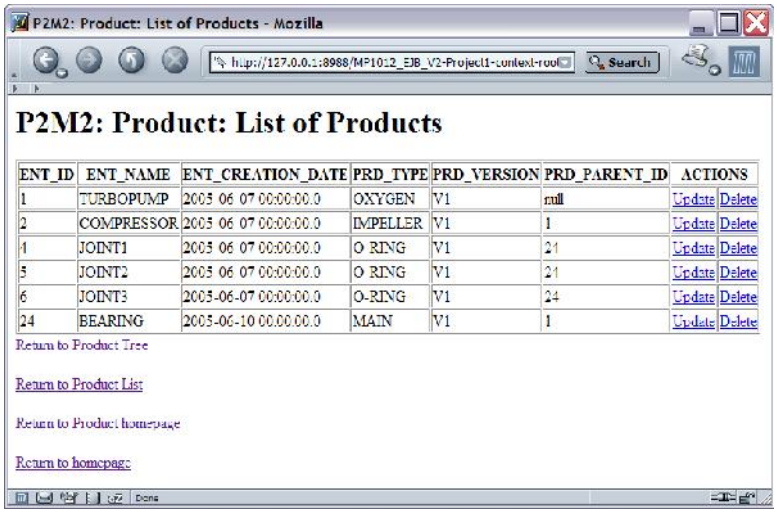

Fig. 8 a List of products.

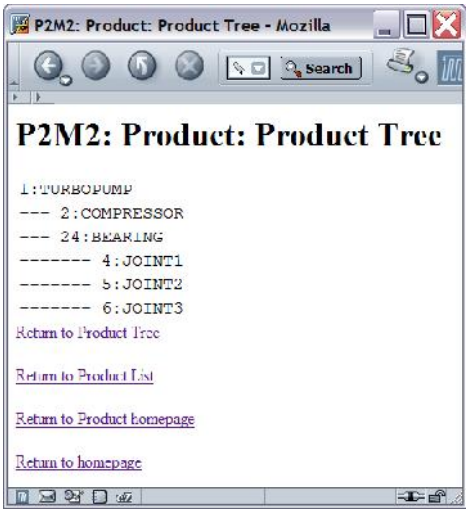

Fig. 8 b products hierarchy in P2M2.

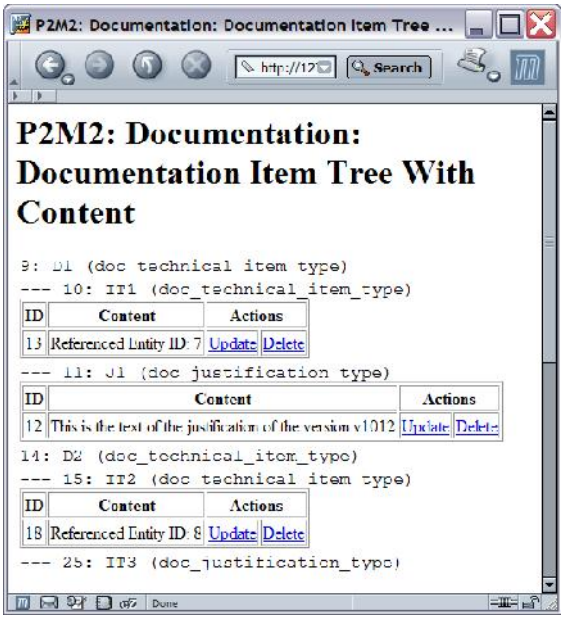

Fig. 9 a Documentation tree

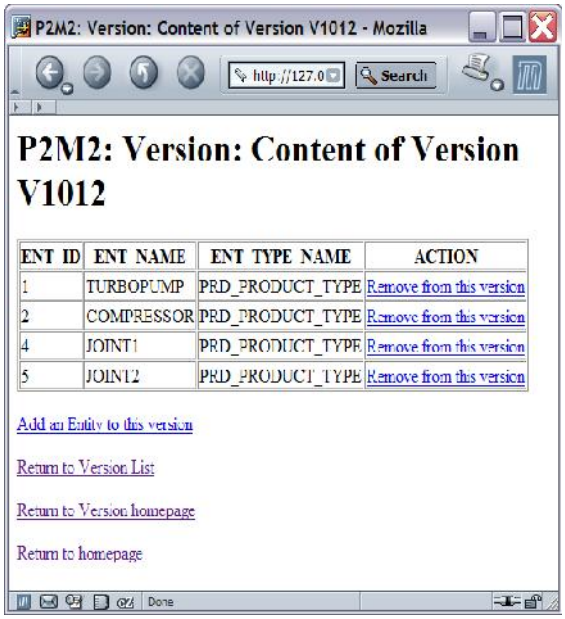

Fig. 9 b Version content

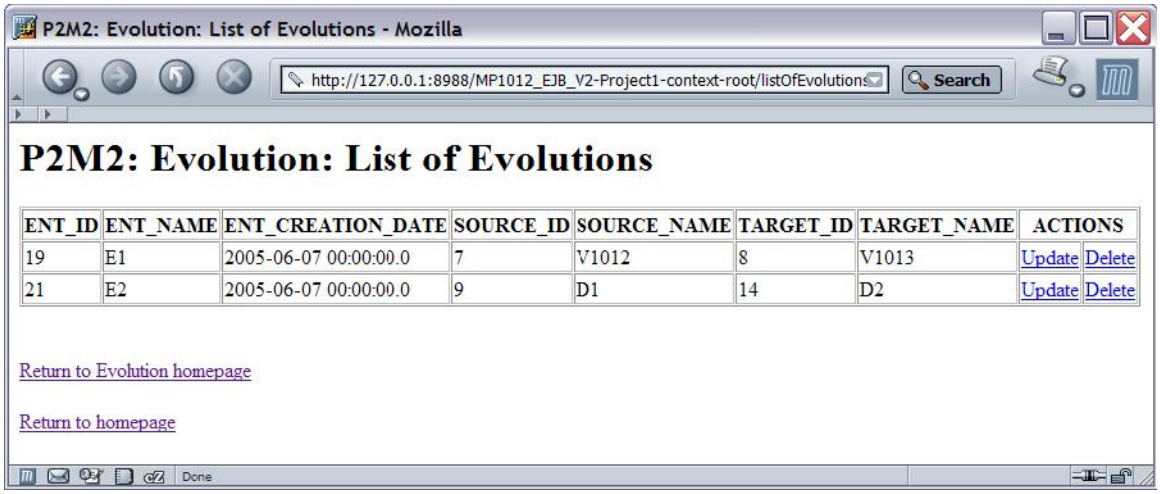

Fig. 10 List of evolutions in P2M2

4. Ontologies for Project Memory Management

4.1 Ontologies and the Semantic Web

Over the past decade, knowledge representation research has focused on ontologies, which are, as [8] defines them, formal specifications of conceptualizations. An Ontology typically consists in a set of formally-defined concepts and properties. By applying inference rules defined by an ontology, a software agent can compute inferred knowledge from a given ontology asserted knowledge. Several general inference tasks are referenced by ontology engineering, the main one being the subsumption problem, which consists in finding all the classes a given object belongs to.
Ontologies usually rely on a subset of first order predicate logic known as Description Logics [1]. Depending on the constructs an ontology is built upon, a given ontology can be proven decidable, in the sense of subsumption, and therefore ensure tractability of applications using it. *SHOIN*(D) is one of the most popular Description Logics, since a well-known implementation of it is OWL-DL, the Description Logic compliant subset of the Web Ontology Language (OWL) [14]. The World Wide Web Consortium (W3C) has recently issued a recommendation for OWL as a core technology of the Semantic Web, but only with its DL restriction can one ensure an EXPTIME decidability regarding the subsumption problem.

Unlike expert systems that aim at building the most extensive knowledge base or ontology pertaining to a specific domain and to compute it, Semantic Web ontologies tend to focus on knowledge interchange and interoperability. The Semantic Web can be regarded as a http-based network of XML-serialized ontologies, such that any resource is identified by an URI (Uniform Resource Identifier, of which URL are a subset). On the Semantic Web, a resource can assert a relation involving some instances of a concept defined by another ontology, simply by referencing the URI of that concept. Therefore, using applications that are aware of a limited set of primitives, a machine can compute knowledge formalized in distinct ontologies, and have access to the meaning of a description rather than to plain text strings.

4.2 Benefits of the OWL-DL data-model over Object-Relational technology

Although Object-Relational data-bases offer a very powerful environment for knowledge management, several insufficiencies remain under this paradigm that an ontology-based approach might help to solve.
The main difference between Object-Oriented modelling, and frame-based systems (built upon Description Logics) comes from the way classes (or types, concepts...) are defined. When creating a new object type in an Oracle application, one can assert that the new type extends any previously defined type, but there is no other way to create a subtype than to explicitly state what type it extends. Under the Description Logics paradigm, a concept is defined by asserting a set of necessary conditions, under which any instance in the knowledge base will be classified as an instance of the considered concepts, whenever those conditions are met. Thus, a knowledge base (KB) has to be classified, and depending on the complexity of the concepts necessary or necessary and sufficient conditions, a software agent, known as a reasoner, will achieve classification of the KB in a tractable time. Therefore, from a set of asserted concepts and roles

definitions, a DL-based KB will compute an inferred ontology, including a concept taxonomy.

For these reasons, ontology-based modelling can be regarded as much more extensible than UML-based models, since whenever a new concept has to be defined, there is no need for a complete redesign of the model.

Under the Semantic Web, one will also benefit from another kind of flexibility due to ontology-based modelling: since concepts can be formally identified by an URI, there is no need to maintain an exhaustive database of any concept one might have to refer to inside the system. By using external URIs, and thanks to Web Services, one can model projects involving externalized resources. Let's suppose a product is completely designed and the model makes use of external references through the URI mechanism. Whenever someone will need a new functionality for which no form was intended (e.g. to check that all the parts of a product, including subcontracted ones, are free of certain particles), the ability to scale a query up to the semantic web will be a powerful advantage.

Conclusion

We have designed P2M2 a generic framework suited for project memory management. We present its platform independent model, and explain its implementation over object-relational technology, in compliance with the model-driven application (MDA) paradigm. We discuss an extension of our work to support ontologies for project memory, and present its benefits. In the near future, we aim at extending P2M2 beyond product centred-aspects of design projects, and to validate our tool in an industrial context.

References

[1] F. Baader, D. Calvanese, D. McGuiness, D. Nardi, "The Description Logics Handbook", Theory Implementations and Applications, Cambridge, 2003.
[2] M. Bigand, "Information system supporting the functional specification in product design", International Federation of Automatic Control 11th IFAC, Symposium on Information control Problems in Manufacturing, INCOM, Salvador, Brasil, 2004.
[3] S. Buckingham Shum, "Negotiating the construction and reconstruction of organizational memories", Journal of Universal Computer Science, 3(8):899-928, 1997.
[4] J.E. Conklin and M.L. Begeman, "gIBIS: A Hypertext Tool for exploratory Policy Discussion", ACM Transactions on Office Informations Systems, 6:303-331, 1998.
[5] R. Dieng, O. Corby, A. Giboin and M. Ribière, "Methods and Tools for Corporate Knowledge Management", International Journal of Human-Computer Studies, 51:567-598, Academic Press, 1999.
[6] J.L. Ermine, M. Chaillot, P. Bigeon, B. Charenton and D. Malavielle, "MKSM a method for knowledge management", Knowledge management: organization, competence and methodology, ISMICK'96, Schreimenmakers ed., Rotterdam (Neederlands), pp.288-302, 1996.
[7] Gamma E., R. Helm, R. Johnson, and J. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", Reading, Addison-Wesley, 1995.
[8] T. R. Gruber, "A translation approach to portable ontologies", Knowledge Acquisition, 5(2):199-220, 1993.
[9] M. Jarke, Experience-based knowledge management: a cooperative information systems perspective, Control Engineering Practice Vol 10 (2002) 561 – 569.
[10] J.J. Kasvi, M. Vartiainen and M. Hailikari, "Managing knowledge and knowledge competences in projects and project organizations", International Journal of Project Management, Pergamon ed., 2003.
[11] M. Klein, "Capturing Design Rationale in Concurrent Engineering Teams", IEEE, Computer Support for Concurrent Engineering, January, 1993.
[12] P.Malvache and P. Prieur, "Mastering Corporate Experience with the REX Method", Management of Industrial and Corporate Memory, ISMICK'03, Compiègne (France), pp.33-41, 1993.
[13]. N. Matta, O. Corby and M.Ribière, "Méthodes de capitalisation de mémoire de projet", INRIA, Rapport de recherche n°3819, 1999.
[14] D. McGuiness, F. VanHermalen, "OWL Web Ontology Language Overview", http://www.w3.org/TR/2004/REC-owl-features-20040210/, W3C recommendation, 2004.
[15] Object Constraint Language Specification. OMG Unified Modeling Language specification version 1.5, http://www.OMG.org/uml, 2003.
[16] OMG-Meta Object Facility Specification v1.4, April 2002.
[17] "Oracle Application Developer's Guide - Object-Relational Features", Release 2 (9.2), Part No. A96594-01, March 2002.
[18] G. Schreiber, H. Akkermans, A. Anjewierden, R. De Hoog, N. Shadbolt, W. Van de Velde, B. Wielinga, "Knowledge engineering and management: The CommonKADS Methodology", The MIT Press ed., ISBN 0-262-19300-0, 1999.