A General Framework for Knowledge Management System Design

Stefania Bandini, Ettore Colombo, Paolo Mereghetti and Fabio Sartori
DISCo - University of Milan - Bicocca
via Bicocca degli Arcimboldi, 8 - 20126 - Milan (Italy)
tel +39 02 64487857 - fax +39 02 64487839
Email:bandini, ettore.colombo, paolo.mereghetti, fabio.sartori@disco.unimib.it

Abstract— **One of the main issues of traditional approaches to Knowledge Based System (KBS) development is related to knowledge and software maintenance. Moreover, knowledge sharing both among different KBS modules and among users is another issue when the KBS technology is exploited for complex software systems. In this paper we propose a framework based on a three–tier architecture that exploits Component–Based approach to overcome these problems and that defines a general framework for the design of Knowledge Management systems. A clear separation between knowledge processing components and other components at the application layer is the central idea of the proposed architectural pattern. The proposed framework has been designed and successfully applied within the context of P–Truck, an ongoing research project that involves the University of Milan–Bicocca and the Business Unit Truck of Pirelli Tyres.**

I. Introduction

Knowledge Management (KM), even if there is not a common agreement within the scientific community [7], can be considered as a process that takes place within an organization. Through this process, organizations generate value from their intellectual and knowledge–based assets [4]. Most often, generating value from such assets requires sharing them among employees, departments and even with other companies in a continuous effort to devise better practices, that are the most important resource for an enterprize to be competitive on global markets. Thus, enterprize interest in supporting KM through the adoption of computer–based tools and methodologies has significantly grown over last years, becoming a trend within the different communities of researchers in Computer Science. Such tools and methodologies can very often be led to the Artificial Intelligence (AI) area [15]. Moreover, within this area, Knowledge Based Systems (KBS) approach is one of the most suitable and diffused when dealing with KM systems [16].

The general framework presented in this paper is dedicated to support KM within organizations whose members belong to particular communities, each one referred to as Community of Core Knowledge Practitioners (CoCKP). A CoCKP is a particular type of Community of Practice [17] that produces and works on the core knowledge of an organization. Due to their wide experience, Core Knowledge Practitioners (CKPs) are able to find innovative solutions to difficult problems (e.g. in product and process design), and thus they play a fundamental role within organizations.

Two of the main open issues related to the development of KBSs concern knowledge maintenance and sharing problems. Hence, defining the framework following a KBS approach, these problems have been tackled. Researches in KBS area have demonstrated that it is possible to supply suitable tools and methods to capture, represent and model different domain knowledge. This is particularly true when such tools have the ambition to improve KM dealing with core knowledge [3]. However, one of the main problems that arises and causes the quality reduction of the knowledge engineering activity is that developed models are often difficult to be updated. This is due because they are generally focused on acquisition, representation and use issues rather than on the maintenance one. KBSs that have been built according to very specific domain core knowledge often require updating or enhancement due to the high level of volatility of such kind of knowledge. Hence, it is fundamental to move from knowledge representation standpoint to a knowledge maintenance one [9]. In order to do this it is possible both to design modifiable knowledge models and to define a software architecture that allows easy knowledge and software maintenance.

Inside a CoCKP, the core knowledge is distributed among members of the community. Nonetheless, as shown in [2], it is possible to represent the core knowledge of the community describing it by a knowledge artifact. Consequently a KBS can be built to solve a problem in which the knowledge involved in the solution is represented according to the obtained knowledge artifact. Hence, the resulting KM system can be viewed as a collection of dedicated KBSs, one for each tackled problem.

Moreover, the decision making process of a CoCKP could require close interactions with other communities within the same enterprize. Since knowledge sharing among CoCKPs is required (i.e. different practitioners of the same community usually have to share knowledge in order to achieve common goals), also the improvement of knowledge sharing among KBSs becomes an important issue to increase the quality of the whole system.

Thus, knowledge sharing, among both different system modules and different users, is an important issue to be considered in addition to knowledge maintenance when KBS technology is exploited for the development of complex software systems. Traditionally, knowledge sharing aspects

are considered marginal by KBS developers: every KBS is devoted to support the solution of a specific problem by the adoption of a well defined problem solving strategy over a knowledge model, structured into a knowledge–base. A complex software system to support CoCKPs, on the other hand, should be designed considering multiple KBSs that work on heterogeneous knowledge, because of the variability of problems tackled by CoCKPs.

Software Engineering area, more than KBS one, is strongly oriented to augment the level of maintenance and distribution of computer systems through the design of modular architectures, in which there is a clear separation between functionalities and data. Thus, the integration of Software Engineering techniques and KBS approach could improve the effectiveness of the latter in supporting complex decisional processes.

The proposed three–tier architecture will be shown in the next section. In Section III a KM system developed according to this architectural pattern is shown as an example of its effectiveness in the design and implementation of real applications. Concluding remarks will end the paper.

II. An Architectural Pattern to Support CoCKPs

The proposed framework is the result of the combination of two very different approaches to the development of KM tools, in order to solve the critical problems of knowledge sharing and maintenance exploiting a modular architecture. The resulting approach is based on an integration between KBS and Component–Based approaches [5].
The proposed solution to tackle knowledge maintenance and sharing issues has been shown in Subsection II-C. Previously, KM system functionalities and the Component–Based approach have been described respectively in Subsections II-A and II-B. Accordance between the proposal and the software engineering approach has been explained in the latter subsection.

A. A Set of Functionalities for KM Systems

To define and better understand the framework presented in this paper, it is necessary to identify a set of functionalities to be implemented building a KM system to support CoCKP work. Such functionalities have been defined following the traditional KBS approach (e.g. [11]):

- *Knowledge Capture*: since a KM system models domain specific knowledge, it should be possible to acquire and to modify this knowledge whenever necessary, providing users with specific tools for the elicitation of knowledge and the maintenance of the developed knowledge base. Due to the extreme variability of their knowledge, this is particularly true when considering CoCKPs;
- *Knowledge Storage*: to save in a persistent way the knowledge about the domain and the problem they are devoted to;
- *Knowledge Deployment*: to provide users and system modules with a sort of controlled accessibility to the stored knowledge;
- *Knowledge Processing*: to support CKPs activities exploiting the knowledge stored and shared in the same system. This functionality refers to the process in which, through inferences based on the stored knowledge and a problem solving method, a KBS finds a qualitative solution to the problem it has been built for.

B. A Component–Based Approach

Software development process defines which activities are important and in which order they have to be carried on in order to manufacture, deploy and maintain software systems. In the last decade the world of software development has rapidly evolved. In particular, the diffusion of computer networks, such as the Internet, has changed the perspective to view the development processes. Moreover, Internet–oriented information systems have been evolving into complex service platforms responding to increasing needs for business service support. Such software systems are perceived as collections of interacting components, instead of as single, often huge, bundles. Component–based reference frameworks have been developed to support the creation of complex business services by assembling a collection of independently developed macro–components within a multi–tier middleware infrastructure.
For the purposes of this paper, we focus on the development of business and middleware component systems, which can be defined as a set of cooperating components assembled to deliver a solution to a business problem [10]. In turn, a business or middleware component is a component that implements a single autonomous business or a middleware concept.
Moreover, an effective component model needs to capture those features that are useful to address system development issues identified in the previous paragraph.

The proposed component model will refer to the following component definition [5]:

1) A component is a self–contained software construct that has a defined use, has a run–time interface, can be autonomously deployed and is built with foreknowledge of a specific component socket;
2) A component socket is software that provides a well–defined and well–known interface to supporting infrastructure into which the component will fit;
3) A component can be built for composition and collaboration with other components.

C. A Pattern Architecture for KM Systems

The main feature of the three–tier architecture (shown in Figure 1) is the clear separation between application components implementing problem solving strategies and the ones providing other services. The aim is to separate functionalities shown in the previous subsection over the different tiers of this framework.
The problem–solving strategy is distributed over a set of Knowledge Processing Modules (KPM), each one devoted to support a CoCKP role. All these modules constitutes one of

the tiers of the framework, referred to as KPM tier. Components belonging to this part of the framework are designed as KBSs. In this way different problem–solving strategies for each specific problem can be chosen according to their effectiveness in representing CKP decision making processes. Furthermore, heterogeneous knowledge–based modules can coexist within the same system even if based on different approaches (e.g. forward chaining production rules, case based reasoning) . Functionalities mainly provided by components of this conceptual tier are knowledge processing and capture.

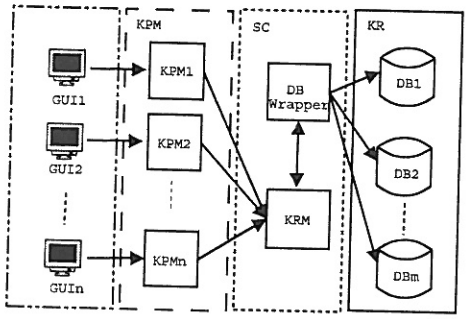

Fig. 1. The proposed three–tier architecture

Another kind of components has been introduced to provide knowledge storage functionality. In fact, the representation of knowledge and information needed by KPM tier is implemented into a Knowledge Repository (KR), that can be viewed as a set of information sources (e.g. databases, XML files). Latter components could be enterprize information system parts or otherwise databases specifically built to support KPM. Since some of these sources are usually heterogeneous and are generally difficult to be accessed due to a lot of causes (i.e. presence of confidential data, security policies, and so on), our framework introduces a specific Middleware Service Component (MSC) tier. The main aim of this category of components is to manage the communication between KPM and KR. According to functionalities previously described, Knowledge Deployment is the functionality assigned to components of this tier.

At this level of the framework it is possible to identify two main components: the Knowledge Repository Manager (KRM) and the DB Wrapper modules. KRM provides every KPM component with a set of functionalities to work on KR (e.g. access, visualization, updating), while the DB Wrapper interacts with the DBMS of the target database. When a KPM asks KRM for operating on KR, its request is interpreted by DB Wrapper, that translates the query, sends it to the DBMS, receives the result, translates and returns it to KRM. In this way, potentially geographically distributed databases can be managed as a single and centralized knowledge source.

From the knowledge maintenance point of view, this architecture allows to provide CKP with a knowledge repository and a flexible KRM tool to manage it. A centralized knowledge base in the KR components facilitates all the activities that require to access it and, at the same time, it is an advantage also from the knowledge sharing point of view. In fact, this solution allows to provide different KPM components with a specific view on the knowledge base that they can access to perform their tasks. Finally, advantages from the software maintenance point of view are provided by the possibility to simply add new KBSs that exploit the same knowledge base content.

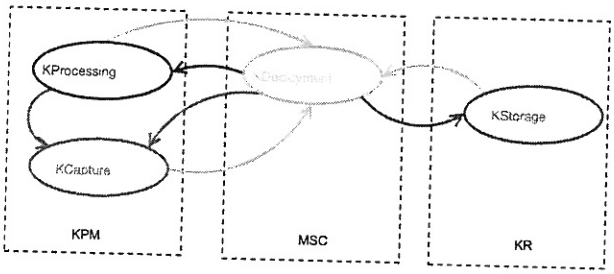

Fig. 2. Functionality Net: how tiers work to provide functionalities

Figure 2 shows how the three tiers exploit their main functionalities to allow other ones to complete and to supply the assigned task. Only through this collaborative way, components can implement such functionalities. Each tier, in fact, needs the support of at least another one to complete its task. In figure, dependencies are indicated using arrows: outgoing ones stand for the given support while ingoing ones indicate the supported functionalities. For instance, *knowledge capture* requires *knowledge processing*, while each functionality can be involved in completing *knowledge deployment*. Since dependencies among tiers need communication among components, both in the same and through different architecture levels, how different components interact is an important issue to be considered in implementing a KM system following the proposed framework.

D. Architectural Remarks

In the previous subsection the three-tier architecture has been introduced and tiers have been described. Existing relations between functionalities and tiers have guided the definition of the proposed architecture. In this section some architectural aspects will be deeper described in order to shown the accordance of the proposed architecture to definitions listed in subsection II-B and to explain some technological choices.

Elements belonging to the defined tiers are components. In fact, ones belonging to the KPM layer can be easily described by the definition 1. Each KPM can be viewed as a software construct offering a service (defined use), having a run–time interface through which it can be accessed by the Graphical User Interface and built with foreknowledge of a specific MSC component socket. In a similar way MSCs can be described as components that define the KPMs supporting infrastructure (definition 2). From a conceptual point of view two or more components, belonging to KPM or MSC layers, can be aggregated in order to build components (definition 3). Moreover, the whole system can be viewed as a component. From the software architecture point of view, there are many frameworks that help to implement the described architectural pattern in a real software system. Enterprise Java Beans

(EJB [14]) and Web Services (WS [12]) are two of the more mature and widely diffused frameworks available nowadays. Both of them are particularly suitable in order to build component–based software architectures because they define standards for component search, service invocation, data exchange and so on.

III. A Framework Application: Supporting Experts in Truck Tyre Design

The presented framework is a significant step in the definition of tools and methods to support CoCKPs. It is general enough to be applied to a lot of domains characterized by heterogeneous knowledge and it allows to overcome problems that must be tackled when a KBS approach is undertaken (i.e. maintenance of the knowledge base and knowledge sharing among experts and integrated KBSs) by adopting an architecture based on the Component Model. A real application of this framework has been implemented in the context of the P–Truck project, an ongoing research project involving the University of Milan–Bicocca and the Business Unit Truck of Pirelli Tyres to realize an integrated KM system to support the design and manufacturing of truck tyres. Knowledge acquisition sessions with Pirelli experts have pointed out that the different phases of the truck tyre life–cycle are accomplished by dedicated CoCKPs, that sometimes interact to solve complex problems. Thus, the P–Truck system has been designed as composed by heterogeneous KBSs, each one dedicated to a specific tyre production phase. In particular, P–Truck knowledge base may require frequent updates and must be shared by all the involved CoCKPs. In the following, the architecture of P–Truck is illustrated in order to explain how it has been designed according to the architectural pattern described in previous sections.

A. Truck Tyre Production Process

The truck tyre production process can be roughly divided into the following phases:

- *Design of rubber compounds*: a rubber compound is a blend of different ingredients, chosen with the goal of achieving required performances, such as tensile strength, resistance to fatigue and so on. The designer decides the composition of the blend, identifying ingredients to be adopted and their amount;
- *Mixing*: raw materials must be suitably mixed in order to obtain a homogeneous blend;
- *Semi–manufactured production*: metallic reinforcements are added to rubber compounds, getting the different parts the tyre will be composed of;
- *Assembly*: semi–manufactured parts are assembled into a semi–finished product, in jargon called *green–tyre*;
- *Vulcanization*: also referred to as Curing, the green tyre is "cooked" in order to give it the required thermal–mechanical features.

Most important phases are *Design of rubber compounds*, *Mixing* and *Vulcanization*, since they provide the final product with all the required thermal–mechanical characteristics. Each of these phases is conducted by a specific CoCKP: Figure 3 briefly shows the truck tyre production process from the CoCKPs' interactions standpoint.

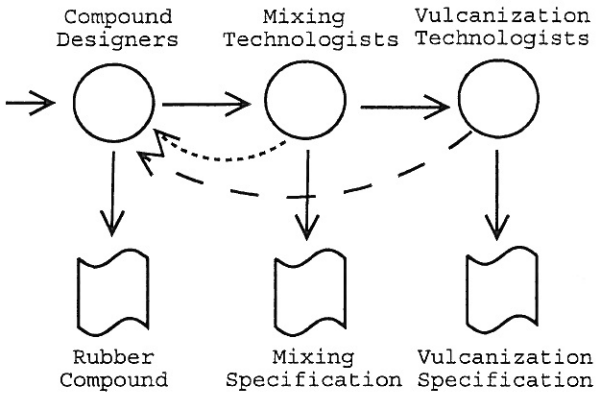

Fig. 3. Interactions among different CoCKPs involved in truck tyre production

Normally, the development of a new truck tyre starts with the definition made by compound designers CoCKP of the ingredients that rubber compounds should contain, in order to satisfy well defined thermal–mechanical requirements. These requirements are suggested by *Marketing* or compound designers CoCKP itself. Then, Mixing technologists design the different steps the mixing process is made up of. Finally, Vulcanization technologists defines a vulcanization process for the green–tyre whose output is the final product. This process, represented by solid arrows in Figure 3, is typically *distributed*: each CoCKP accomplishes its own activity exploiting its own expertise and no interactions with other CoCKPs is established.

Anyway, there are some situations in which different CoCKPs have to interact in order to solve common problems. Such situations are usually unpredictable. In particular, two important kinds of such interactions have been identified and highlighted in Figure 3: *compound designers–mixing technologists* (i.e. the dotted arrow) and *compound designers–vulcanization technologists* (i.e. the dashed arrow). These interactions represent the negotiation process among Pirelli CoCKPs. For instance, a negotiation can start when mixing and vulcanization technologists cannot accomplish their activity due to some kind of anomaly referrable to rubber compound design (e.g. not availability of one or more ingredients considered in a blend compound recipe).

B. P–Truck Architecture

The architecture of P–Truck, shown in Figure 4, has been designed according to the architectural pattern described earlier, in order to have a *centralized knowledge repository* and a *distributed problem solving strategy*. As it will be better explained later, the designed architecture provides a lot of benefits, in particular, from the knowledge maintenance and sharing viewpoints.

The problem solving strategy is implemented by KPM tier components of the system, that is made up of four Knowledge

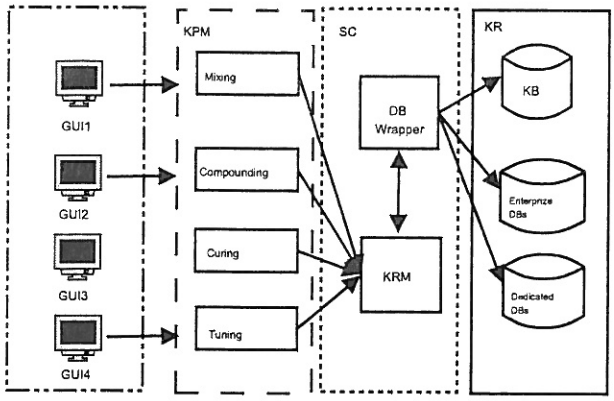

Fig. 4. P–Truck Modules

Based Systems that are: *Compounding* (supporting compound designers), *Mixing* (supporting mixing technologists), *Curing* (supporting vulcanization technologists) and *Tuning*. The aim of the latter KBS is to handle possible anomalies that may occur during the production phases. Members of the supported CoCKP come from manufacturing process design CoCKPs and, when needed, they negotiate to solve the encountered anomaly.

The user can interact with a KPM exploiting a specific GUI. It has been preferred to create dedicated GUIs in a common Web environment rather than a unique one, in order to satisfy the requirements of every CoCKP.

Each KPM module is devoted to reproduce the problem solving strategy of a specific CoCKP. In this way, it has been possible to choose different approaches for managing different kinds of involved knowledge. In particular:

- Compounding and Mixing have been developed as production rule–based systems. The main reason for this is that through the knowledge acquisition campaign models of the involved knowledge have been defined and the solution methods adopted solving these problems are properly represented by sets of rules;
- Curing and Tuning have been designed as case–based systems, since the vulcanization step and the solution of process anomalies are characterized by heterogeneous and episodic knowledge, that can be appropriately captured by the CBR paradigm [6]. See [8] and [1] for more detailed descriptions of these modules.

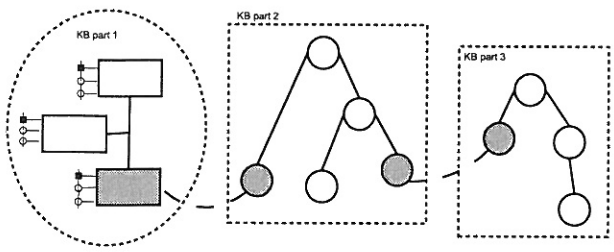

Fig. 5. The Knowledge Base of P–Truck

Another important system tier is the KR, made up of a centralized knowledge–base (KB) and a collection of enterprize databases. This tier contains all the knowledge and data necessary to single KPM elements to work properly. In particular the knowledge base is structured according to defined knowledge artifacts [2].

In order to properly support these heterogeneous KPMs, P–Truck KB has been internally partitioned into different parts belonging to two logical type, as shown in Figure 5:

- tree–structured KB, that allows a flexible management of the knowledge necessary to P–Truck Curing and Tuning;
- relational KB, drawn according to the classical ER model implementing the defined knowledge artifacts, modelling knowledge involved in Compounding and Mixing.

KB parts are not functionally separated but are linked by means of boundary elements that belong to connected knowledge models. In Figure 5 boundary elements are indicated by the gray filled shapes and linked by a dashed line. Two linked boundary elements, belonging to two different KB parts, represent the same conceptual entity of the domain. In this way, possibly heterogeneous models developed for distinct KPMs can be shared, reproducing typical knowledge overlapping among CoCKPs. For example, knowledge that allows to give a tree–structure to a blend recipe (otherwise a flat list of couples ingredient–amount) is shared by all the experts of truck business unit of Pirelli.

Middleware Service Component, the last important tier of P–Truck, contains two important modules: the Knowledge Repository Manager (KRM) and the DB Wrapper. The Knowledge Repository Manager (KRM), that provides users and KRM allows system users (i.e. P–Truck users and the KPM modules) to manage the KR content, providing them with functionalities like visualization, deletion, retrieval and update of the stored knowledge. Figure 6 shows a screenshot of the graphical interface of the P–Truck module that allows users to visualize KR contents. This graphical representation reproduces the internal structure of the KR that users can navigate to consult the KR content.

General information (e.g. about ingredients to be included in rubber compounds, machineries to be used in Mixing and Curing phases) contained in Pirelli's databases are not replicated in the P–Truck System. In fact, an opportune service component has been developed in order to extract this information directly from enterprize DBMS when required (i.e. the DBWrapper). This module acts as a configurable interface between KRM and enterprize databases. The DB Wrapper has been developed in order to provide the KR with possibly heterogeneous and geographically distributed data that are already available in the enterprize information system. Wrapped enterprize data are organized and structured within the KR according to expert knowledge and integrated with other useful data. For instance information about ingredient properties are structured in a way that takes into account their relationship to blend and tyre features. The DB Wrapper has been designed and developed in such a way that allows it to be independent from the nature of information sources: when a KPM module (or a P–Truck user) needs some data, its request is sent to the DB Wrapper.

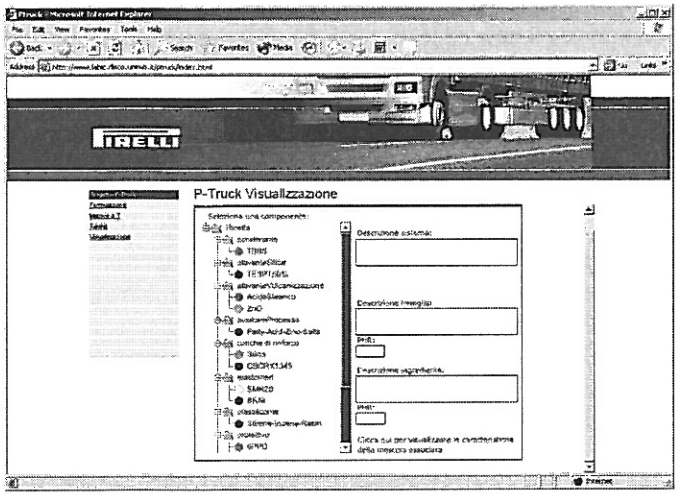

Fig. 6. An example of GUI: a structured recipe visualization

The latter translates the query into a suitable format for the information source that contains required data (e.g. an Oracle Database), executes the query and returns the result into a suitable format for the requiring KPM (or the user interface).

IV. Conclusion

This paper has presented a framework for the development of complex knowledge–based systems based on the component paradigm. The adoption of software engineering techniques is very useful when designing KBSs: in fact, while traditional methodologies for the construction of KBSs are heavily focused on how modelling domain knowledge to reproduce experts' problem solving methods, they usually ignore other aspects that are close to the software engineering area. In particular, maintenance and sharing are two important issues in the design of complex KBSs, especially when the knowledge involved in the problem to be represented and solved is heterogeneous. This is the typical situation when KM systems must support Communities of Practice and, among these, Communities of Core Knowledge Practitioners. A system to support CoCKPs cannot simply take care of building a good knowledge representation, but it must consider that this knowledge should be continuously maintained and shared among all the community members.

On the first issue, it is possible to distinguish between two different types of maintenance. In fact, knowledge maintenance could be presented as a particular case of software maintenance or as an interaction between the system and the CoCKP that must update the knowledge stored in the system. By defining the framework according to a software engineering approach, software maintenance can be made easier and improved. For what concerns the knowledge maintenance, the implementation of knowledge capture, deployment and storage functionalities as particular components of the framework allows CoCKPs to manage knowledge previously modelled.

About the knowledge sharing issue, the general framework presented in this paper guarantees knowledge sharing among software modules and among different CoCKPs. The proposed solution involves deployment and storage functionalities. The results of knowledge processing operation could be considered either deployed by the system or stored in the system by CoCKP members. In this way, complex KM systems are able to share knowledge both directly inserted by CoCKPs and resulting from the use of the implemented problem–solving strategy.

References

[1] S. Bandini, E. Colombo, F. Sartori and G. Vizzari, *Case Based Reasoning and Production Process Design: the Case of P-Truck Curing*, European Conference on CBR 2004 (ECCBR04), Madrid. In press.

[2] S. Bandini, E. Colombo, G. Colombo, F. Sartori and C. Simone, *The Role of Knowledge Artifacts in Innovation Management: the Case of a Chemical Compound Designer CoP*, International Conference on Communities and Technologies (C&T 2003), pp. 327-345. Kluwer Academic Publishers, Dordrecht, 2003.

[3] S. Bandini and S. Manzoni, *Modeling Core Knowledge and Practices in a Computational Approach to Innovation Process*, in L. Magnani, N. Nersessian (eds.), Model-Based Reasoning: Science, Technology, Values, pp. 369-390, Kluwer Academic/Plenum Publishers, New York, 2002.

[4] T. Davenport, L. Prusack, *Working Knowledge - How Organizations Manage What They Know*, HBS Press, 1998.

[5] P. Herzum and O. Sims, *Business Component Factory*, OMG Press, Wiley, 2000.

[6] J. Kolodner, *Case Based Reasoning*, Morgan Kaufmann Pu., San Mateo (CA), 1993.

[7] C. Lueg, *Knowledge Management and Information Technology: Relationship and Perspective*, Upgrade - Knowledge Management and Information Technology, edited by C. Lueg, X. Alamain. Available on www.upgrade-cepis.org. Vol. 3, No. 1, 2002, pp.4–7.

[8] S. Manzoni and P. Mereghetti. *A tree structured case base for the system ptruck tuning*. In UK CBR workshop at ES 2002, Cambridge, 10 December 2002, pp. 17–26, Glasgow, 2002. University of Paisley.

[9] T. Menzies,*Knowledge Maintenance: The State of the Art*, The Knowledge Engineering Review, Vol. 14, No.1, 1999, pp. 1-46.

[10] M. Morisio, C. Seaman,V. Basili, A. Parra, S. Kraft and S. Condon, *COTS–Based Software Development: Processes and Open Issues*, Journal of Systems and Software, December 2001.

[11] A. Preece, D.H. Sleeman et al., *Better Knowledge Management through Knowledge Engineering*, IEEE Intelligent Systems, Vol. 16, No. 1, January/February 2001, pp. 36–43.

[12] E. Newcomer, *Understanding Web Services: XML, WSDL, SOAP, and UDDI*, Eddison–Wesley, 2002. ISBN: 0201750813.

[13] J. Robertson, 2002, *Benefits of a KM Framework*, *Intranet Journal*, available at http://www.intranetjournal.com/articles/200207/*pse*_07_31_02*a.html*

[14] E. Roman, S. Ambler and T. Jewell, *Mastering Enterprise Java Beans*, John Wiley & Sons, 2002. ISBN: 0–471–41711-4.

[15] M. Santosus and J. Surmacz, *The ABCs of Knowledge Management*, available at http://www.cio.com/research/knowledge/edit/kmabcs.html, 2002.

[16] E. Tsui, B.J. Garner and S. Staab, *The role of artificial intelligence in knowledge management*, Knowledge Based Systems, Vol. 13, No. 5, October 2000, pp. 235-239.

[17] E. Wenger, *Community of Practice: Learning, meaning and identity*, Cambridge University Press, Cambridge, MA, 1998.