Published in: Smari, W.W.; Melab, N.; Yetongnon, K. (Eds.): Proceedings of the International Symposium on Information Systems and Engineering, ISE'2001, Las Vegas: CSREA Press 2001, pp. 114-121

Knowledge Management Systems: Essential Requirements and Generic Design Patterns

Ulrich Frank
Institute for Information Systems Research
University of Koblenz
Koblenz, Germany

Abstract

In recent years, Knowledge Management Systems (KMS) have drawn remarkable attention. However, there is no common understanding of how a knowledge management system should look like or where the corresponding research should be directed at. Based on a number of essential requirements, a KMS should satisfy, this paper introduces a generic architecture for KMS. It consists of a conceptual framework that suggests to structure knowledge according to three perspectives and five aspects. To support the differentiation of common, generic knowledge, domain specific knowledge, and information, the architecture features three levels of abstraction, namely an ontology level layer, a domain level layer and an operational level layer. Unfortunately, the conceptual representation and implementation of these layers faces a number of challenges. The paper presents and discusses three alternative design patterns to overcome these challenges.

1. Introduction

With the growing popularity of knowledge management, it does not come as a surprise that an increasing number of software vendors claim their systems would qualify as Knowledge Management Systems (KMS). Searching the web for the term 'Knowledge Management System' retrieves more than 10.000 pages. However, apparently there is no unified interpretation of the term. Instead, most of the software featured under this label turns out to belong to traditional types of systems, such as text retrieval systems, document management systems, systems for customer relationship management, database management systems, help desk systems, teachware or hypermedia systems. It seems that many vendors regard knowledge management as a marketing tool that emphasizes certain aspects of system deployment rather than as a specific technology. Research on knowledge management systems is characterized by a remarkable diversity, too. While everybody seems to agree that knowledge management systems are safeguards to retain mission critical knowledge and that they are essential to avoid the erosion of organizational knowledge, there is no common understanding of how a knowledge management system should look like or where the corresponding research should be directed. Work on knowledge management systems takes place in various disciplines. Often it originates in systems that had been developed earlier under other labels. In Artificial Intelligence, knowledge based systems or tools to develop and maintain ontologies are related to knowledge management ([1], [2]). In Information Systems Research, knowledge management serves as a vehicle to re-invigorate decision support systems [3]. Data Warehouses as knowledge repositories are subject of research on information systems, data base systems and knowledge based systems ([4], [5], [6]). The term 'Organizational Memory System' is used in various disciplines. However, usually the descriptions of organizational memory systems remain on a vague level or are essentially characterized as hypertext or hypermedia systems (for instance: [7]).

Against this background the question 'what is a knowledge management system anyway?' might be regarded as redundant: apparently there are different interpretations of the term in various scientific communities and therefore it is hardly possible to resolve its ambiguity - similar to the term 'information system' - by introducing a unified concept. However, most of the existing systems that are related to knowledge management have been originally designed with other objectives than knowledge management. The approach we suggest here is different in the sense that it starts with a reflection on the notion of knowledge - and on generic requirements a system that is to manage knowledge should fulfil. To satisfy these requirements, we will outline a generic architecture of knowledge management systems that supports the acquisition, representation, maintenance and dissemination of knowledge on various levels of abstraction.

In order to implement this architecture, one can fall back on a large range of proven concepts from software engineering in general and from conceptual modelling in particular. However, at the same time, knowledge management demonstrates a series of additional, distinctive requirements that bring with them attractive research problems. The paper will introduce and compare three design patterns that are suited to meet these challenges.

2. Knowledge Management Systems: Requirements and Features

Any attempt to define the term knowledge has to face a dilemma. On the one hand, knowledge - both as part of colloquial and scientific language - seems to be a self-evident term with no need for further explanation. Nevertheless, you can find many deviating definitions of knowledge. That makes it almost impossible to find a definition that is compatible with most existing notions of knowledge. On the other hand, knowledge represents a phenomenon that is very difficult to reflect upon. While we can speak about knowledge, any insight into knowledge can be regarded as knowledge itself. Similar to language we can differentiate between knowledge and knowledge *about* knowledge (meta knowledge). But although we can do that over many levels, in the end we cannot avoid a *regressum ad infinitum*. For these reasons, it seems to be a frustrating endeavour to develop a comprehensive definition of knowledge. Fortunately, such a definition is not necessary for our purpose. We mainly need a pragmatic image of knowledge that is suited to differentiate KMS from traditional information systems.

2.1 Knowledge versus Information

With a purely formal approach, as it is common in computer science, it is not possible to distinguish knowledge from information. Knowledge is not a central term of semiotics either. However, semiotics offers concepts that help with a distinction between data and information. Taking into account the differentiation between syntax, semantics and pragmatics [8], data can be regarded as symbols that adhere to a specific syntax. Syntax and formal semantics (like the operational semantics defined for an Integer in a programming language) are defined with a corresponding type or schema. Information on the other hand depends on the relationship between data, human perception and action. If data represent objects in the world view of a person that are suited to influence his judgement and action, we can regard it as information. Information content grows with the number of possible interpretations it excludes. In philosophy, knowledge is essentially related to cognition, intellectual discovery, explanation and understanding. Therefore there is emphasis on methods to structure, give reasons for and evaluate scientific knowledge. In other words, the philosophical notion of knowledge stresses originality, abstraction and reason. How does this relate to information? Whether or not information qualifies as knowledge depends on its originality, abstraction and reason - aspects whose judgment clearly vary with the human interpreter. However, abstraction can be specified more precisely than the other aspects. Abstraction implies not to refer to single objects but to types, classes or concepts.

Our brief excursion to semiotics and philosophy - although it would deserve a much more thorough analysis - gives us some hints how to characterize knowledge in the context of this paper. First, knowledge does not denote single objects but always classes or concepts. Second, knowledge should be convincing, i.e. it should be accompanied by proper explanations. Third, knowledge should be suited to help the stakeholders of a business firm with understanding, analyzing and eventually changing its strategy, organization, and core business processes.

2.2 Requirements

In general, a KMS should serve everybody who is involved in processes of understanding, evaluating and (re-) organizing the business. Due to the nature of these tasks, the primary focus groups include consultants (internal and external), new employees who have to understand the company in general and their task in particular, executives, system analysts as well as customers and suppliers that participate in cross-organizational business processes. A KMS should provide these groups with relevant knowledge. With respect to the notion of knowledge introduced before we can now describe more specific requirements, a KMS should fulfill.

Emphasis on Concepts and Reason

A KMS should offer definitions of concepts that are needed for the description and analysis of a corporation. To give a few examples

for such concepts: corporate strategy, organizational unit, business process, task, employee etc. Note that these concepts are usually not defined independently from one another. In contrast to a traditional information system, a KMS should allow to answer questions that refer to concepts, for instance:

- What is a business process?
- What are the concepts to describe a new type of business process?
- What are the organizational implications of a strategy that is aimed at cost leadership?

Re-use of Existing Knowledge

Although there is no unified terminology for the description of corporate knowledge, there are a number of elaborated and well documented concepts available - provided, for instance, by text books. This is also the case for the documentation of relevant causal relationships. A knowledge management system should provide an adequate body of existing knowledge. This is for various reasons. The re-use of knowledge does not only contribute to the economics of a KMS. It should also improve the overall quality of its content. In addition to that it fosters communication by referring to a body of knowledge many people are familiar with.

Support of Multiple Perspectives

In order to support different users and different tasks, a KMS should provide various perspectives on the knowledge it stores. Managing complexity recommends offering different levels of detail. For instance, sometimes it will be sufficient to get a description of a business process that is restricted to an outline of the temporal relationships between high level tasks. In other cases it may be important to provide a comprehensive description of every task within the process as well as of the required resources. The plethora of intellectual tasks to be performed in an organization is usually accompanied by a separation of concerns. Classes of problems are related to certain professional communities. In order to support these communities, a KMS should provide concepts that relate to corresponding specialized languages and abstractions.

Integration with Information

With a KMS there is emphasis on storing knowledge rather than information. However, information cannot be completely neglected. This is for two reasons. Firstly, the distinction between knowledge and information depends in part on subjective judgement (see 2.1). For this reason the complete exclusion of information is not possible. Secondly: While it is obvious that knowledge adds value to related information, it is also the case that information adds value to knowledge. Since knowledge is rather abstract, having access to corresponding instances (that would be regarded as information in the light of the terminological discussion above) will help many people to develop a proper understanding. Therefore a KMS should support the integration of knowledge with information.

Support of Awareness

In order to foster organizational learning, a KMS should support the dissemination of knowledge. Whenever its content gets updated, users that are interested in the corresponding topics should be notified. For this purpose a user should be able to subscribe to certain types of knowledge or - more general - of content.

3. A Generic Architecture for KMS

At first sight, it may seem that the requirements presented above could be fulfilled by a hypermedia system. However, the requirements have some implications that demand a more specific architecture. Firstly, a KMS should allow for storing formalized knowledge. *Formalization* improves a system's integrity (see 2.1). It also fosters the integration of knowledge with information, because a precise, formal specification of knowledge allows to define semantic relationships to information (for instance through instantiation associations). Furthermore, it provides for the implementation of powerful notification mechanisms: a user can subscribe to precisely defined classes of knowledge. Secondly, as a consequence of storing formalized knowledge, a KMS has to feature a formal *language* that allows for describing relevant knowledge in an appropriate way. In order to guide the user with representing knowledge - and with searching for it, there is need for structure. For this purpose we suggest a general framework to structure knowledge in an enterprise using well known abstractions. It is inspired by a method for multi-perspective enterprise modelling (MEMO, [9]). MEMO differentiates three so called *perspec-*

tives - strategy, organization and information system - each of which is structured by four *aspects*: structure, process, resources, goals. From a user's point of view, the framework presented in fig. 1 can be regarded as the "main menu" of the system that offers different foci to "zoom" into. The various foci (a focus is a particular aspect within a certain perspective) within this framework are illustrated by a few characteristic terms. In addition to this framework, the requirements suggest to differentiate between generic knowledge (that is valid for a wide range of companies), domain specific knowledge (that is valid for a smaller set of companies) and information that is used on an operational level. The three level architecture depicted in fig. 1 reflects these thoughts. The top layer serves to store and manage concepts of specialized languages. We call it the *ontology level layer*. The second layer, which we call the *domain level layer*, stores descriptions of knowledge that result from applying the concepts defined in the top level layer. Finally, the *operational level layer* stores information that is related to domain level descriptions.

An architecture that implements these three layers would perfectly fulfill the requirements. On each layer the basic framework would be used to structure knowledge or information respectively.

The architecture promotes the re-use of generic knowledge in terms of specialized terminologies, which correspond to typical text book knowledge. Also, it allows to re-use domain specific knowledge, for instance reference models of business processes or corporate strategies for particular types of business firms. Users can subscribe to content they are interested in by referring to changes of concepts - either on the ontology level layer or the domain level layer.

4. Specific Design Patterns

So far, the proposed architecture remains on a rather abstract level. Its implementation requires a more concrete specification, which faces a number of severe problems. As will be shown, the levels of abstraction offered by common object-oriented programming languages are not sufficient to directly reproduce the layers of the architecture in fig. 1. Another, more subtle problem results from the specific shortcomings of instantiation relationships. The generic design patterns presented in the following sections show different approaches to deal with these problems. Notice that we do not apply a comprehensive structure (like the one suggested by [10]) to document a design pattern. Instead, our focus is mainly on the concepts and their implementation (corresponding to "structure", "consequences" and "implementation" in [10]).

4.1 Design #1: Emphasis on Instantiation

To define the representation of the ontology level layer, we fell back to the specification of a set of modelling languages we designed previously as part of MEMO. It includes graphical languages, like an organisation modelling language, a strategy modelling language or an object-oriented information modelling language

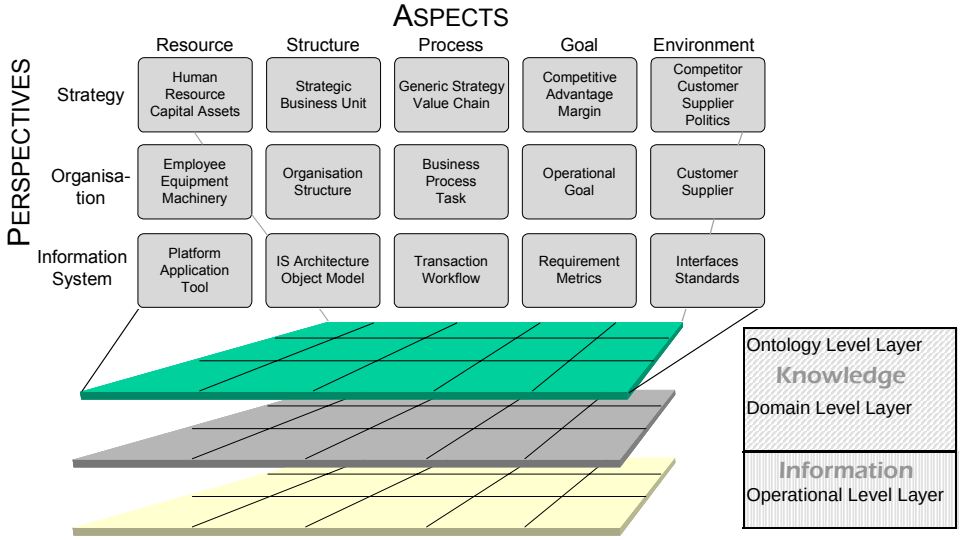

Fig. 1: Basic Framework and Levels of Abstraction

[9]. The languages are specified with metamodels. Meta models can easily be reproduced by object models.

The following approach to design a multi level KMS as it is outlined in 3 corresponds to a design known from CASE tools. The static aspects of the ontology level layer are designed by an object model that represents a meta model, which defines a particular modelling language. Hence, implementation of this layer would result from implementing each class in the object model through a corresponding class in the implementation level language. The concepts used on the domain level layer are then instances of corresponding concepts on the ontology level layer. For example: The specification of a particular process type, such as "Order Management", would be an instance of the class ComplexProcessType. It would be linked to instances of other classes that are associated with ComplexProcessType. It would be straightforward to define a particular process as an instance of a corresponding object on the domain level layer. However, this is not possible with common object-oriented programming languages. Usually, those languages offer two levels of abstraction: classes and objects (instances). Apparently these two levels are not enough for our purpose, since we would need to create an instance from an instance. To overcome this difficulty, the artefacts needed on the operational level layer can be generated from the object on the domain level layer - similar to the generation of code from an object model.

While this design satisfies in part the requirements discussed above, it has a number of disadvantages:

Conceptual redundancy: Generating concepts on the operational level layer from concepts on the domain level layer results in redundancy. For instance: A particular process type that is represented by an object on the domain level layer would be transformed into a class on the operational level layer. Only then, an instance of this generated class would be used to represent a particular instance of the process type. Assuming that generation does not imply the loss of semantics, this would mean that there are two representations with the same meaning.

Limited integration: Generating artefacts implies to loose their integration with the source. For instance, changing the generated artefacts could happen independently of the source thereby compromising the system's integrity.

Limited re-use: It is the basic idea of the ontology level layer to provide the user with a professional terminology that incorporates all the knowledge that is relevant within a certain per-

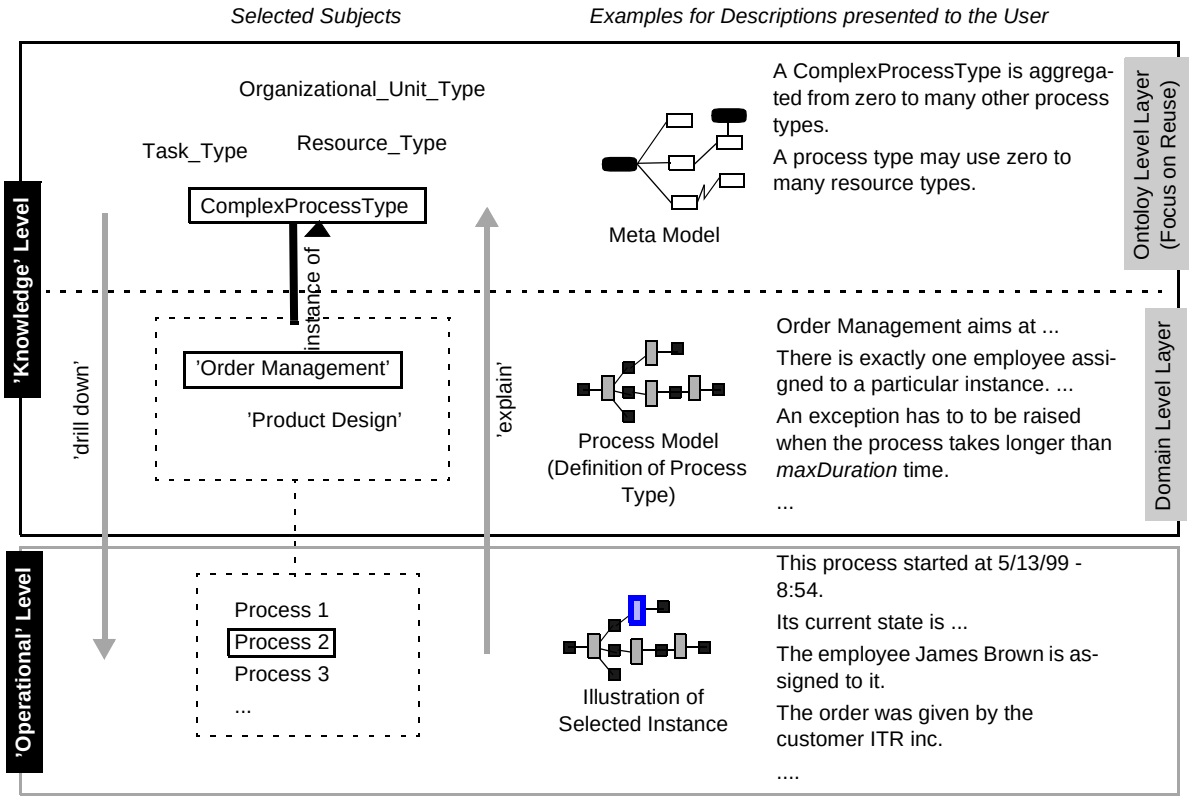

Fig. 2: Levels of Abstraction within Design Pattern #1

spective on enterprises. However, due to the abstraction on the ontology level layer, the knowledge that is expressed here relates to concepts (or types) only - although there might be generic knowledge that applies to all instances of a type, too. But within this design there is no way to express features of instances. For example, it is well known that each process has a start time and may be an end time. These would be typical attributes of a class that represents a process type. But here, the attributes of a class on the ontology level layer define features of process types - and a process type does not have a start time or and end time.

4.2 Design #2: Emphasis on Inheritance

This alternative is a response to disadvantages encountered with design pattern #1 - especially the limited re-usability of generic knowledge. To overcome these problems, design pattern #2 features inheritance relationships between the ontology level layer and the domain level layer rather than instantiation relationships. This decision has a clear impact on the content of the ontology level layer: It will not contain the terms of a language anymore, instead it will provide generic concepts, which can be specialized into domain specific concepts. At first sight, the resulting design seems almost the same as the one in fig. 2. However, from a logical point of view, featuring inheritance relationships between the ontology level layer and the domain level layer implies a fundamental difference to the use of instantiation relationships. Nevertheless, the distinction between both types of relationships is not intuitive because of their ambiguous use in natural language: we use the same denominator - "is a" - for both relationships. For instance: "Order management is a process" where "a process may contain other processes" or: "Order management for hazardous goods is an order management process." In both cases "is a" can either mean that a process (type) is being specialized from an existing one or that a process (type) is one possible specification out of many types that are instances of a meta process type. In the case of an instantiation relationship, the concepts on the ontology level layer are used to define process types. Hence, they describe features of types only. This is different with inheritance relationships. In this case, the ontology level layer would contain generic concepts, like a generic business process type, which could be specialized into domain specific process types. Hence, a generic type could include all the features every instance of any possible specialized type should have. Despite this advantage, this pattern implies some severe disadvantages, too:

Lack of Flexibility: The domain level is restricted to types that are specialized from those on the ontology level layer. If there is need for additional types - in other words: for additional terms on the domain level layer, a modification of the ontology level layer will be the only option. Changing the ontology level layer, however, compromises the chances to re-use the content of the domain level layer in other environments. Additionally, different from design pattern #2, it is not possible to express features that relate to the set of instances that belong to a type. For example, one could not express a feature like 'averageDuration' of all processes of a certain type.

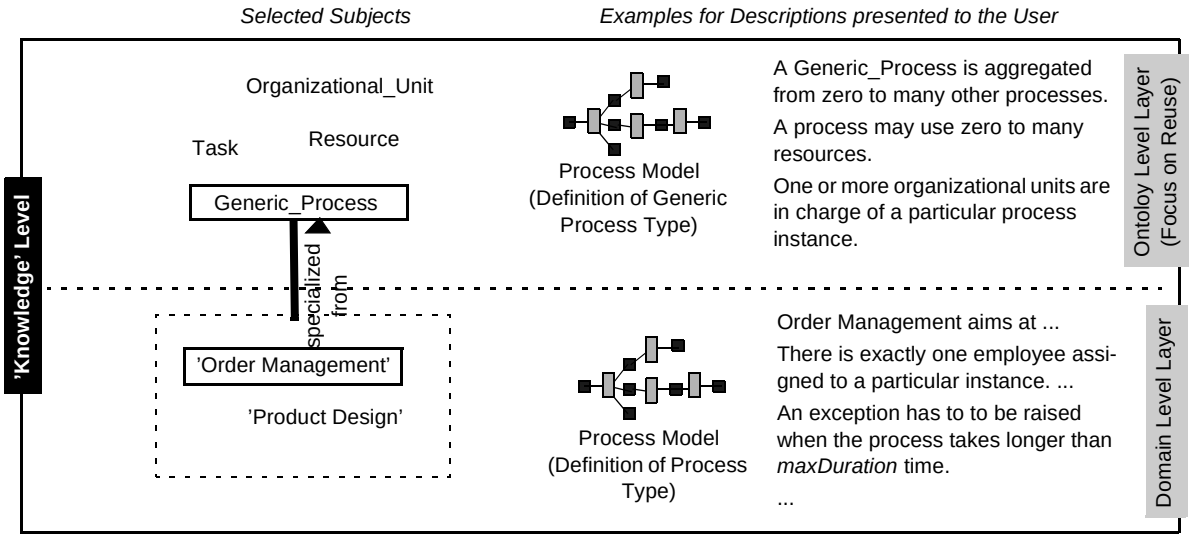

Fig. 3: Design Pattern #2: Emphasis on Specialization

Lack of satisfactory Specialization Semantics: The specification of domain level concepts depends on the chances to find proper specialization from generic concepts on the ontology level layer. Unfortunately, the specialization of concepts faces a number of serious challenges. Firstly, for complex concepts, like a process type, it is hard to find a convincing specification of specialization relationships. For some process types, it seems to be obvious that they are related via a specialization relationship. However, formalization requires a precise definition of specialisation. For instance: Is a process type a specialization of an existing one, if it contains additional subprocesses? As an orientation, any proposition that is made about a general concept should hold for the specialized concept, too. Consider the specialization of generic process type, e.g. Generic_Process (see fig. 3). If a specialized type, like Order_Management required a specialization of Role, for instance a Department_Manager, it might seem appropriate to specialize Department_Manager from Role. However, that would result in a problem that is similar to the notorious covariance problem known in object-oriented software engineering [11]: The proposition that every instance of a Generic_Process (or one of its subtypes respectively) may be associated with an instance of Role would not hold anymore for Order_Management, because an instance of Order_Management would require a specific subtype. In other words: such a covariant redefinition of features would result in a logical contradiction.

To summarize, design pattern #2 is an option only, if the domains that are subject of a KMS are relatively stable and the designers of a system are confident to have a comprehensive understanding of these domains.

4.3 Design #3: Additional Metaclasses

The two design patterns considered so far come with disadvantages that can hardly be accepted in many cases. An ideal solution would combine the expressive power of instantiation relationships with that of inheritance relationships. Our investigations in this requirement indicate that there is no such ideal solution. The only way we found to specify an alternative that is superior to the ones described above is to apply a "conceptual trick". It consists of introducing an additional level of abstraction, namely metaclasses. A metaclass serves to describe features of a class, which is its only instance. Hence, it is possible to express that ‘number of instances’ or ‘average Duration’ are features of a set of process instances (represented by a class) which are stored with a class (see fig. 4). The instance of a metaclass is initialised on the layer the class belongs to.

The implementation of metaclasses depends on the concepts provided by the implementation level languages. Implementation is convenient with languages like Smalltalk that feature metaclasses. Otherwise one might define special

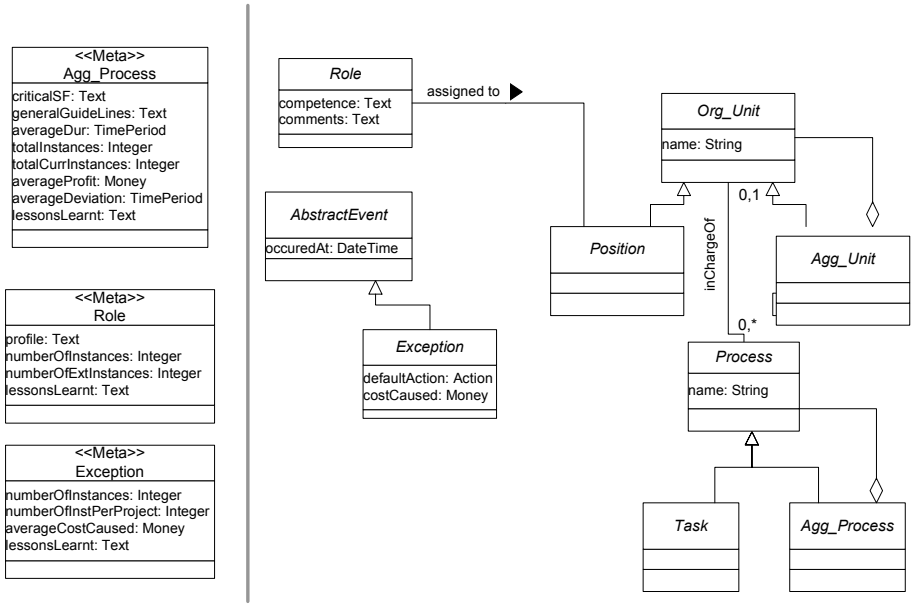

Fig. 4: Additional Metaclasses to represent Type Features (Design Pattern #3)

classes with a sole instance that serves to represent features of a class. In addition to that one has to protect the semantic integrity of the relationship between a virtual metaclass and the corresponding class.

5. Conclusions and Future Work

Against the background of essential requirements suggested for KMS, we introduced a three level architecture of KMS. The conceptual specification and even more so the implementation of these layers faces a number of subtle problems. Design pattern #1 is suited only if there is no need to store knowledge/information about types (or classes respectively). On the other hand, design pattern #2 is not suited for dynamic domains where one has to face frequent changes of generic knowledge. Design pattern #3 is a compromise that combines the advantages of the two other patterns. Unfortunately, the introduction of metaclasses implies additional difficulties with those programming languages that do not feature metaclasses. Our future research on KMS will take two parallel directions. To evaluate the usability of the concepts on the ontology and domain level layer, we deploy corresponding conceptual models in projects we currently conduct with a number of business firms. First results indicate that an intuitive graphical representation is of pivotal importance for gaining acceptance with users. However, as soon as the models become subject of detailed analysis, the benefits of precise (formal) definitions become evident. The other direction of our future research aims at investigating alternative ways to represent knowledge. This includes the use of knowledge representation languages from Artificial Intelligence as well as corresponding inference engines. An additional "meta ontology" layer would allow for gaining even more flexibility. It would feature a meta language that could be used to specify additional special purpose languages - or to modify/enhance an existing language. For this purpose we plan to use an already existing meta-metamodel [12].

References

[1] Farquhar, A.; Fikes, R.; Rice, J.: The Ontolingua Server: A Tool for Collaborative Ontology Construction. In: International Journal of Human-Computer Studies, 46(6): 707--728, 1995

[2] O'Leary, D. E.: Using AI in Knowledge Management: Knowledge Bases and Ontologies. In: IEEE Intelligent Systems, 13(3)34--39, 1998

[3] Holsapple, C. W.; Whinston, A. B.: Decision Support Systems: A Knowledge-Based Approach. Course Technologies 1996

[4] Erdmann, M.: The Data Warehouse as a Means to Support Knowledge Management. In: (Ed.): Proceedings of the 21st Annual German Conference on AI '97: Freiburg, Germany, September 9th - 12th (http://www.dfki.uni-kl.de/~aabecker/Freiburg/Final/ws-ki-97-proceedings.html). 1997

[5] Inmon, W. H.: Building the Data Warehouse. 2. Ed., Wiley 1996

[6] Gray, P.; Watson, H. J.: Decision Support in the Data Warehouse. Prentice Hall 1998

[7] Euzenat, J.: Corporate Memory Through Cooperative Creation of Knowledge Bases and Hyperdocuments. In: (Ed.): Proceedings 10th Banff Workshop on Knowledge Acquisition for Knowledge-Based Systems. SDRG Publications 1996, pp. 1-18

[8] Morris, C. W.: Writings on the General Theory of Sign. Mouton 1971

[9] Frank, U.: Visual Languages for Enterprise Modelling. Arbeitsberichte des Instituts für Wirtschaftsinformatik. Arbeitsberichte des Institut für Wirtschaftsinformatik der Universität Koblenz-Landau, No. 18, 1999

[10] Gamma, E.; Helm, R.; Johnson, R. E.; Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley 1998

[11] Meyer, B.: Object-oriented Software Construction. 2. Ed., Prentice Hall 1997

[12] Frank, U.: The MEMO Meta-Metamodel. Arbeitsberichte des Institut für Wirtschaftsinformatik der Universität Koblenz-Landau, No. 9, 1998