INTEGRATING MODELS OF QUALITY MANAGEMENT METHODS BY AN OBJECT-ORIENTED REPOSITORY

Peter Peters, Peter Szczurko
Informatik V
RWTH Aachen
52056 Aachen
Germany

ABSTRACT

This paper[1] deals with an integration approach of quality management methods using the deductive object-oriented DBMS ConceptBase as a repository. Based on the IRDS standard we defined a distributed modeling environment. The experiences we had during the modeling process in the research project WibQuS[2] emphasize the need for object-oriented abstraction mechanisms and deductive reasoning. Furthermore the developed conceptual model is a framework not only for specification and evolution of method models but also for information browsing in a distributed information system.

INTRODUCTION

During the last twenty years, quality control has become one of the major research topics in engineering and business disciplines. The paradigm in this area has changed from product inspection to the idea of company-wide *Total Quality Control* (Feigenbaum 1991) managing the whole company in a quality-oriented way. A lot of methods[3] and tools (Quality Function Deployment (QFD), Failure mode and effect analysis (FMEA), Statistical Process Control (SPC), etc.) have been developed and adapted to ensure product and process quality along the product life cycle which form a *quality life cycle* (Pfeifer, 1993). Because of the rising interest in quality control a variety of research groups has been founded, each of them focusing on one specific task or method. On the other hand, the enormous amount of work on CIM indicated that only the integration of methods and a well-organized information flow between them enables correct and efficient work (Scheer, 1988).

Following these requirements, the WibQuS research group (Jarke et al., 1993 b), a consortium of six German engineering departments, a computer science group and a work sciences group investigated an environment for quality management. WibQuS was founded to integrate several quality control methods by building a federated information system to support various quality control tasks. The overall goal of this information system is to support and organize the information flow both between agents executing quality control methods and between knowledge based tools supporting them.

[1] This work was supported in part by the BMFT project WibQuS and by the Deutsche Forschungsgemeinschaft through Graduiertenkolleg Informatik und Technik.

[2] WibQuS stands for “Wissensbasierte Systeme in der Qualitaetssicherung” (knowledge-based systems in quality management).

[3] The term “method” is meant here in a more general way than this is normally done in the OO-community. It describes a general plan or representation an agent uses to solve a class of problems.

From the information technology and database point of view, we have to overcome two major problems to ensure a straight, goal-oriented development of an information system that supports quality management:

1. **The spatial distance**. Not only the project groups, but also the departments of a company involved in quality management are spatially distributed. Furthermore the need of a *distributed information system architecture* and a *distributed development process* is justified by the fact that it is neither practicable nor useful to hold the large amount of data in a central database.
2. **The conceptual distance**. The independent evolution of research groups in quality control has led to different domain languages. Therefore, as a starting point for further system specification and integration, a common conceptual model has to be established. As a consequence, we distinguish between two phases: the method modeling phase, where concepts used by the methods had to be defined, and a system development phase, where the information system and the knowledge based tools will be specified and implemented.

A broader view on method integration embodies not only technical objects but also organizational structures and human communication. Following that, the classical data dictionary approach was enlarged by the idea of repositories which store not only data models, but also objects relevant for development, use and maintenance of data (Habermann, Leymann, 1993). Furthermore, repositories became more interesting because the focus of attention changed from merely *capturing* data to the *sharing and integration* of data (Rose et al., 1992).

We will argue that object-oriented and deductive features are necessary to support repository based model development by describing and analyzing the WibQuS modeling process. While object-orientation is needed to provide necessary abstractions that help to organize and integrate method models, deduction is needed to reason about relations and connections between models in large scale repositories.

In the first part of this paper we describe the special requirements in the WibQuS research group and we will explain how they can be met by a repository embedded in an object-based manager like ConceptBase (Jarke et al., 1993 a). The second part includes the structure of the distributed modeling process. We briefly discuss our experiences with this process model and the problems which occurred. Some remarks about process improvement and the further use of the repository in method support will conclude this paper.

OBJECT-ORIENTED REPOSITORIES IN WIBQUS

As mentioned above, we have to consider the distributed character of the modeling process in quality management. Obviously all the engineers have acquired a large amount of data and experience in their own research area which has to be included into the conceptual model. Therefore, the development process is the determination and formalization of related objects in the world of product, process and method models which is usually lengthy and complicated e.g.

CIM-OSA (Kosanke, Vernadat 1992), STEP (Wenzel, 1991). Furthermore, tool support should consider necessary cooperation and collaboration tasks during the modeling process.

According to the distances described and the complexity of the methods in quality control, the modeling in group sessions is not appropriate. Besides, the engineering groups are used to specifying their models with different modeling languages (e.g. ER, SADT, OO-design, KADS) and they do not have any experience in method modeling. Therefore, a distributed modeling process is more efficient though some additional problems arise during model integration (see section 5).

A convenient support of the modeling process can be given by a tool which enables adequate 'Computer Supported Cooperative Work'- and method description techniques (Jarke et al., 1993 b). In contrast to the features of existing modeling and specification tools we require the following properties considering the characteristics of the distributed modeling process.

1. Though the modeling of methods will be performed locally, we have to support the integration aspects of the modeling process. That implies the necessity for a general modeling language and the possibility of successive model integration. Automatic detection of inconsistencies and incoherences during the development of method models would be beneficial because of the rising complexity of the conceptual model.
2. The necessity for communication between participating groups is conspicuous due to the detection of interfaces between the models and the definition of interchange knowledge. A tool should provide help in connecting models and guiding the communication process.
3. In spite of the rather decentralized development process, we have to manage the model versions under central control to avoid inconsistencies and to provide an appropriate version control during the whole modeling process.

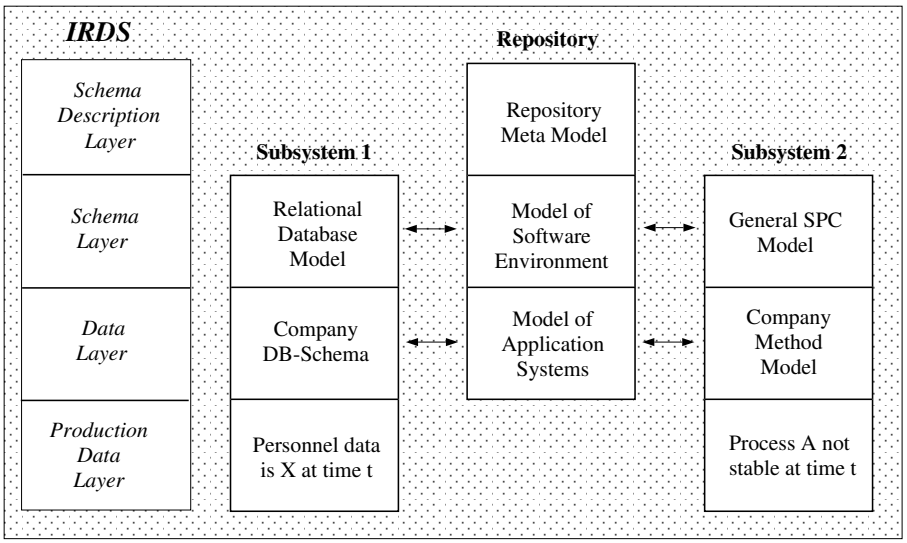

Figure 1: Repository support for layered design environments

To solve the different problems of model integration and intelligent cooperations, a coherent conceptual, technical, and social framework is required. The framework we are going to advocate here is based on repository notions. More precisely, we shall start from the Information Resource Dictionary Standard (IRDS) as recently defined by ISO (IRDS Rapporteur Group, 1993) and shall demonstrate how it can be used and specialized to integrate method models. The IRDS standard is organized by level of classification (c. f. figure 1, Smolander et al., 1991), i.e., level n+1 constitutes a type system for the information described at level n.

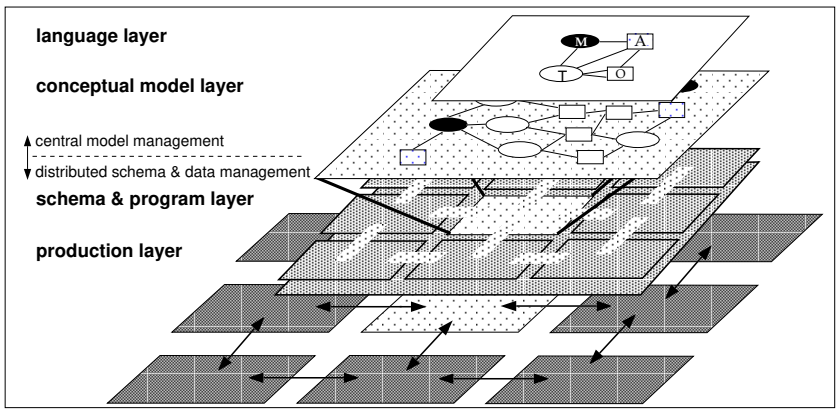

Figure 2: The four layer architecture of the WibQuS repository

The subsystems to be integrated (databases, programs) exist at the level of database state or execution traces, their schema is defined at the second level, and they typically have a hard coded meta level that implements the data model or programming language use. The repository manages schema-level information on the subsystems to be integrated. Therefore, the schemata of the subsystems are (part of) its data; formal and explicit descriptions of the languages used constitute the schema, and we need a repository meta level in which these schema definitions can be described.

A similar four-layered repository architecture was developed in the WibQuS research group which provides a framework for modeling quality control methods (figure 2).

The uppermost layer of the WibQuS repository can be seen as the *language layer* consisting of categories and relations regarded as classes of objects that provide an abstract description of objects and data in the quality management process (figure 6). This layer is the starting point and guide for the definition of the conceptual model that describes the method.

Using the language layer the individual methods have to be formalized on the *conceptual model layer*. Object orientation enables the development of a method network where objects and relations are described as instances of the given classes. While the language layer was defined by the whole WibQuS research group we argued for the distributed development of each method model but also for a central model management by a persistent object memory.

On the *schema and program layer*, the method models are adapted to a concrete manufacturing application. On theone hand that implies the central management of global product and process information and on the other hand the distributed evolution of method specific schemata. For example, the specifications of vacuum cleaner components are stored in a central database while the result specifications of the quality control methods referring to the vacuum cleaner are described by schemata stored in private databases. The communication between the distributed method applications will be organized by the conceptual model.

The data resulting from real processes and methods are stored on the *production layer* as instances of the schemata described at the above layer. These are the data available for information exchange between the participating methods and for verification of the quality management process along the quality life cycle.

Regarding the content of the layers, this repository has been developed quite analogously to the IRDS architecture. Because the description of the methods mostly started from scratch, we could focus not only on *coupling* the methods by describing interface objects, but even more on *developing* method models, bearing the existence of other methods and of information channels between them in mind.

EMBEDDING REPOSITORIES IN CONCEPTBASE

Starting from the repository structure evolved in section 2 we will briefly discuss how it can be implemented by a deductive object base manager like ConceptBase (Jarke et al., 1993 a). We will also give some examples of the ConceptBase modeling language Telos that provides the formalization of deductive and object oriented data models (Mylopoulos et al., 1990).

The deductive and object oriented features support the realization of repositories in ConceptBase in several ways. Telos supports meta modeling by an extensible object structuring mechanism, combined with assertion facilities (rules and constraints) to define the semantics of language extensions. Telos achieves this by a mapping to the perfect model semantics of standard deductive databases (Jeusfeld, 1992). These features enable the definition of an application specific modeling language at the language layer of the repository. The modelers need not define their models in the basic Telos language but can refer to the more familiar terms of the language layer.

The required architecture for repositories can be provided using the following structural facilities of Telos:

- In a repository we need a conceptual abstraction property for the connection of the different layers. We use *instance-of* relations between objects at different layers to express *classification*. In our architecture every task in the conceptual model is an instance of the category Task at the language layer (c. f. figure 3).
- Within a layer classes with the same characteristics can be structured by *generalization* or refined by *specialization*.

- Relationships among objects can be described using *aggregation*. The relations between classes as well as simple value type assignments are described as attributes.

Figure 3 shows the textual notation of an object of the language layer (see also figure 6). The conceptual object *Task* is described as an instance of the predefined Telos language construct *MetaClass* to express its membership of the language layer. Obviously every object which relates to *Task* as an attribute is also instance of *MetaClass*. The facility to define rules and constraints within the *Task* frame enables enriching model semantics. In our example the rule provides the *transitivity* of the include-attribute and the constraint forbids the *circularity* of inclusions. These additional semantics are inherited to all instances of *Task*.

```
Individual Task in MetaClass with
   attribute
      takes : object;
      produces : object;
      includes : Task;
   rule
      transitive : $ forall a,b,z/Task
                     (a includes z) and
                     (z includes b) ==>
                     (a includes b) $
   constraint
      noncircular : $ forall a,b/task
                       (a includes b)
                       ==> not (b includes a) $
end
```

Figure 3: Telos frame of the object Task in the language layer

According to the repository requirements, a repository environment must support three different interpretations of figure 1:

First, the repository must serve as a *model integrator* that supports the definition of static and dynamic interrelations between methods. This is ensured by the client-server architecture which allows on the one hand distributed modeling and on the other provides central model management and analysis. Furthermore, the explicit connection of models and modeling language by instantiation ensures the similarity of the method models and facilitates the detection of connection points between them (see section 4 and 5).

The ability to specify application-specific rules and integrity constraints is another useful property of Telos which facilitates developing semantically correct repository models. As an example we can derive relations from given information and prohibit circular relations which are not significant for the repository architecture.

Second, the repository is also a *communication medium* through which agents exchange requests, data and services. This has influenced the definition and implementation of query classes, an extended view concept supported by ConceptBase. Query classes can also be defined to analyze models for certain desirable and undesirable properties, and to provide simplified views of complex models in large data bases (c.f. figure 4).

Similar to rules and constraints, queries are deductive statements. The ConceptBase query language CBQL represents queries as classes whose instances are the answer objects of the query. Query classes define necessary and sufficient membership conditions for their instances. In our repository approach we use queries as retrieval assistants which derive interesting and critical objects from the conceptual model. These objects can be analyzed for further structuring and agreement in group sessions. Therefore queries guide the communication process by the automatic detection of interesting objects.[4]

For example there might be accepted and rejected annotations to any concepts in the model. These objects show the need for further agreement. While the query (Contradictable) is formulated in terms of the language layer the instances (AnalysingMethod) are elements of the conceptual model.

```
QueryClass Contradictable isA
ConceptualCategory with
   comment, attribute
      explanation : "Objects which are both
                     accepted and rejected
                     by different partners"
   query_rule, attribute
      qr_contra   : $(exists a1 : Agent
                     (this accepted_by a1)) and
                     (exists a2 : Agent
                     (this rejected_by a2)) $
end

Individual AnalysingMethod in Method,
Contradictable with
   attribute
      defined_by
         definer : VK
      not_understood_by
         uncertain : RG
      accepted_by
         ok : SH
      neglected_by
         not_ok : RF
end
```

Figure 4: Query to find contradictable votings

Third, the repository must be a *DBMS* with all database facilities such as efficiency and security. This interpretation led to a number of novel performance optimization techniques which need not concern us here but are of prime importance for system acceptance. Furthermore, ConceptBase is based on a wide-area client-server architecture. Clients and servers run as independent processes which interact via inter-process communication (IPC) channel. A nationwide and even transatlantic collaboration of clients on a common server has been facilitated in various applications (Jarke, 1993). The ConceptBase clients provide tools which allow the representation of objects by graphical semantic nets as well as in textual frames. The ConceptBase server manages and controls the knowledge providing various routines that are carried out when messages arrive from clients. The routines could be *tell* which adds new information into

[4] What was interesting or critical in the WibQuS modeling process will be described in section 4 and 5 in more detail.

the knowledge base, an *ask* which requests one or more instances of a specified query, *browsing* routines (textual, graphical, hierarchical) which provide an overview of the knowledge base and *administrative* routines which are responsible for connection and disconnections of clients and server.

Coupling the presented features with a client-server architecture, a deductive object base manager supports the distributed modeling process in a proper manner. It can be used as a repository store with the properties described above, namely a modeling language, a persistent storage management, communication facilities and the graphical interaction environment in which simultaneous development processes are supported over large distances. Experiences from several applications in software and information systems design (Jarke, Pohl, 1992) show that repositories can be implemented efficiently using this framework.

DISTRIBUTED MODELING IN THE WIBQUS PROJECT

The differing domain languages in the engineering groups was the reason for the development of a conceptual model which could serve as a common language for further tool specification and as a framework for the quality information system. Furthermore, it was helpful to build this model because the precise definitions and potential interactions of quality control methods were initially poorly understood, which complicated communication between the different specialists involved. As a consequence, method modeling was not only intended to agree on a common system specification but also to change the view from *method oriented* to *integration oriented* .

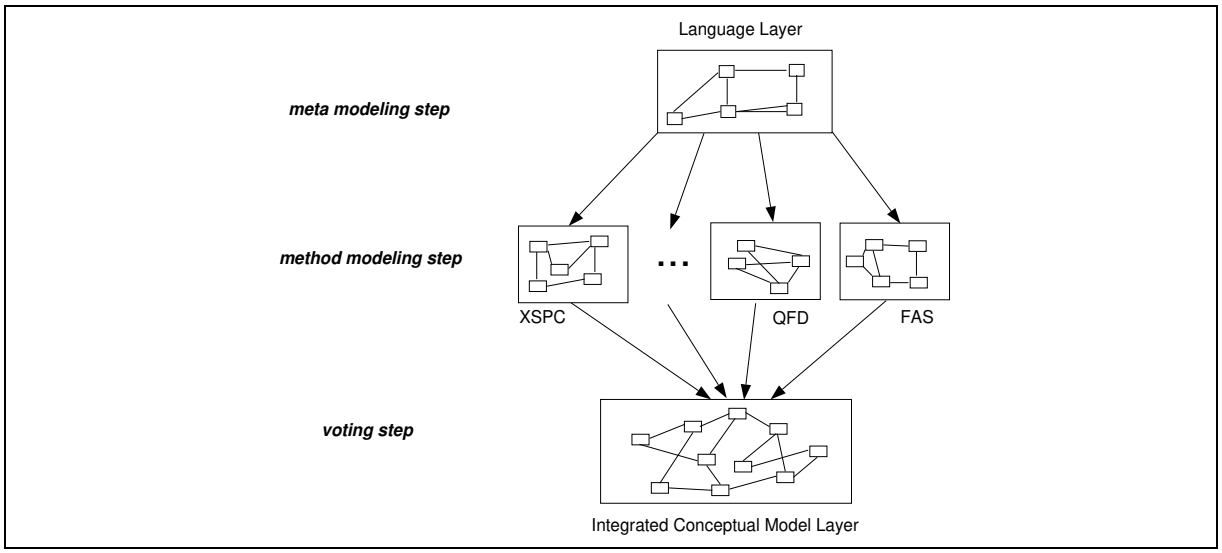

Figure 5: The WibQuS modeling process

Based on the proposed repository structure, the modeling process in the WibQuS research group followed a three-step process model resulting from the requirements of the engineers and the characteristics of the repository approach (c. f. figure 5). In the process the two uppermost levels of the repository structure are considered because according to figure 2 only the concepts of common interest need be modeled.

The goal of the *meta modeling step* was the Definition of the language layer serving as a "modeling language" corresponding to the repository meta model. As described in section 2 the model provided the basic categories for the definition of method descriptions and thus ensured a similar structure for every model.

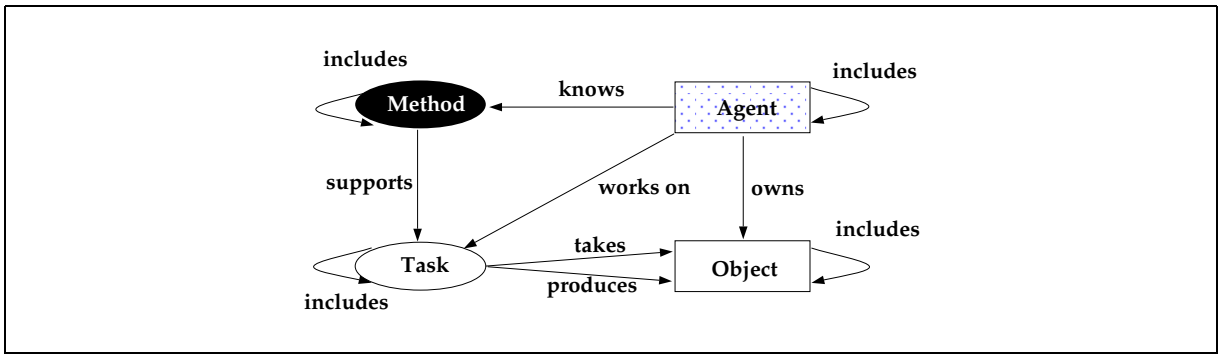

Figure 6: The WibQuS language layer (Jarke et al., 1993 b)

Figure 6 shows a definition of this model in the semantic network syntax of Telos. The relations between the given categories represent dependencies between them as well as part-of relations if one object includes objects of the same category. For example, an assembly task can be divided into subtasks. In every subtask objects can be seen as elements (*Task takes Object*) of the product which will be composed in the manufacturing process (*Task produces Object*). So as a major advantage of the repository approach task decompositions and team structures can be related directly to the modeled methods.

After an agreement on the defined categories and links, in the *method modeling step* every engineering group began to evolve their method models on the conceptual model layer according to the categories of the language layer. Because of the inheritance facilities between the different levels of modeling in the repository, modeling guidance was offered by checking consistency between instance and class. For example, if a task *DetermineDesignElements* is given which has a *takes*-attribute the types of the referring objects are checked.

In the *voting and consistency step* the different models have been connected and problems in concept understanding have been tackled. Connecting was supported by identifying superfluous concepts (e.g. Objects neither taken nor produced by a task), standardizing names of objects modeled by more than one group (and therefore possible interface objects) and distinguishing identically named concepts with different meanings. Understanding was facilitated by making comments on concepts (as in the query answer in figure 4) so the clarification can be done using direct communication between the commenting and the modeling engineer. The cycle of voting and changing concepts had been iterated several times before the conceptual model of WibQuS was formally accepted.

In this last step the deductive facilities like query classes simplified several checks of consistency and coherence using the additional model semantics. The voting on concepts was supported

by the assignment of special attributes (*accepted_by*, *rejected_by*, *not_understood_by* and *comment*) to the concepts of the language layer. These agreement attributes were inherited to every element of the conceptual model. Because of the large conceptual model an evaluation of the voting was only manageable using query classes. The critical elements detected by queries like the one in figure 4 were discussed in a group session.

As a result of the voting step, a structure of integrated method models has been achieved. The repository, until now acting only as a model integrator, in the following can be used for further specification of interfaces between the methods or other method supporting purposes (see section 6).

RIGHT FIRST TIME?

As pointed out in the previous section, the modeling process can be facilitated and accelerated by using a repository embedded in a client-server architecture. The WibQuS research group showed that a large integrated model of seven quality management methods and their methodical environment including about 600 objects can be planned, built and accepted in approximately half a year.

Interviewing the engineers after the modeling phase, it turned out that the object-orientation and the database facilities allowed quick design and analysis of method models. The meta model provided guidance for model structuring and therefore simplified model integration, because it ensured a similarity in structure. The query mechanisms offered fast model checking mechanisms searching for unconnected concepts or for concepts modeled twice. From the technical point of view the model integrator task has been solved by ConceptBase, though there were problems in working with the provided features, because most of them did not appear in the "modeling world" of the engineers before.

The communication medium task arose mainly during the voting step. The table below shows how often the several voting attributes attached to every concept were instantiated in the first voting. Like in the example in figure 4, Concepts were voted and probably additionally commented if they were interesting for the voting group because of model connections or ambiguous meaning. So, all concepts were normally voted more than once. The numbers in the table have also been computed by using special queries defined for this purpose.

While two thirds of the voted concepts have been accepted by the voters, one third has been voted differently or has been rejected. Those concepts have been changed using the comments or have been discussed during meetings between voting and defining group. According to the opinion of the involved engineers, together with the possibilities of electronic mail, the communication medium task has been facilitated by the repository as well, because all of them admitted that they couldn't handle the fairly complex and large model any more without the query support.

Number of objects	592
accepted_by	195
rejected_by	2
not_understood_by	13
comment	178
accepted_by and rejected_by	90
accepted_by and not_understood_by	7

Table 1: The Voting on concepts in WibQuS

Problems which lengthened the process arose with respect to three main aspects:

1. the duration and granularity of the modeling steps,
2. the acceptance of the selected modeling tool and
3. the capacities of the nationwide data exchange networks

While the last point is beyond the scope of our possibilities[5] (but nevertheless very strongly influenced the acceptance of the client-server architecture and therefore the DBMS task of ConceptBase), the second one is a problem one has to be aware of in distributed working. Because of different backgrounds in the engineering groups, a variety of techniques and tools for modeling and specification was established in these groups (SADT, OO-Design, KADS,...). Naturally, these tools influence the engineers' way of thinking about modeling. The conceptual modeling paradigm represented by ConceptBase/Telos forced them to look at their world from another viewpoint and it took time to get used to this viewpoint.

Facing this problem, it is very important to explain possibilities of the technique and features of the tool to people clearly and comprehensively. We were not the first who had the experience that it is the motivation of the modelers that determines if the model turns out to be "good" or "bad". Understanding the tool is one major point that influences this motivation. Therefore we think that a good tool and paradigm tutorial and documentation is an indispensable part of project preparation under the circumstances of distributed modeling.

[5] Because of the nationwide project, we had to use Internet via the net of the Deutsche Bundespost and therefore the unpredictable transfer rates caused regular breakdowns of the system or unacceptable answer times. Due to this, most of the engineers sent ascii—files with the defined models via e-mail and we integrated them into the repository.

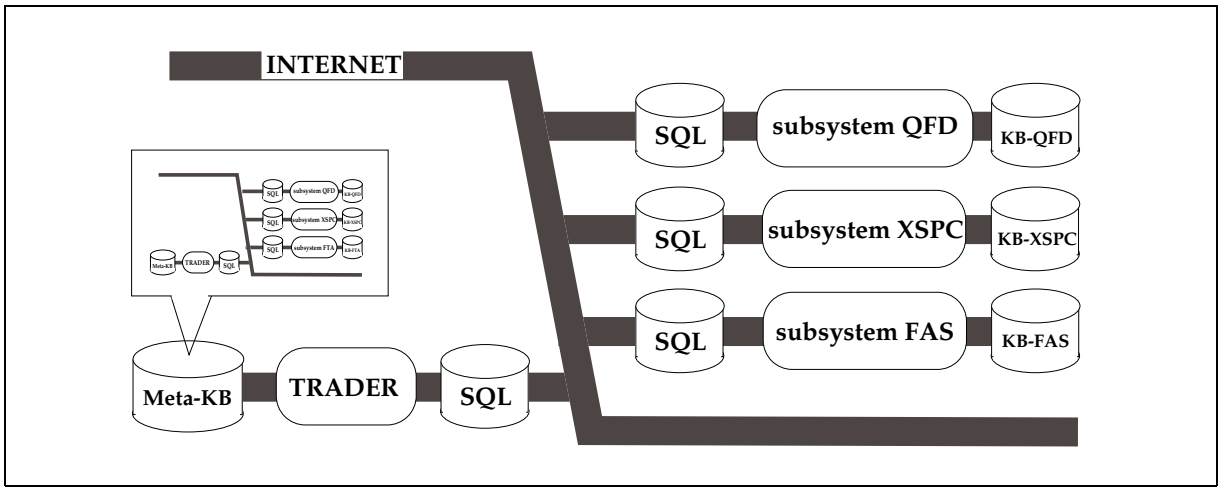

Figure 7: Architecture of the WibQuS information system.

But looking at the modeling process from the integration point of view, the problem of duration and granularity of the modeling steps is the one of main interest. Some trade-offs have to be named with respect to duration and granularity of the process steps. As said before, the model at the language layer is a necessary prerequisite for distributed modeling. Process analysis showed that a small, one-layered model like the WibQuS model provided too much modeling freedom to the engineering groups. As a consequence, models differed not only in structure but also in intention. While some groups tried to represent the structure of the method, others built their models with a software tool already in mind. Resulting in models of different granularity and different vocabulary, this lengthened the voting step. Most of the modelers complained that more time should have been spent on a more detailed language layer in order to provide a better guidance through the method modeling process.

Conversely, a too detailed meta model would hinder the modelers because they could not match their domain languages with the defined categories. Therefore, the language layer should be refined by all project members as long as all of them understand it and agree with it. We paid for a short first meta modeling step by having a very long voting and integration phase. This is quite analogous to the domain we tried to model: a fault detected in the quality planning phase costs a fraction of one detected in the quality checking phase.

REPOSITORIES IN METHOD SUPPORT

While the previous chapters described that repositories provide an environment for distributed modeling of methods, we now present some ideas about how the developed models will be used in communication support. As mentioned above, the modeling phase was only the first step on the way to an integrated quality information system. At the end of this step, the conceptual model now contains all relevant method concepts which will be used in the second step for organization of communication between methods.

Figure 7 represents the proposed decentralized architecture of the quality information system. The knowledge-based systems built by the engineering groups possess their own "private" knowledge bases containing the knowledge relevant only for this specific method (for instance, the set of rules of the XSPC). The interface to the system will be realized by a SQL-database. It contains exchangeable data, described in terms of specializations of the concept Object and building the schema and program layer, and special relations for the cooperation with other subsystems.

The trader will have the role of an exchange coordinator for queries and orders. Its knowledge base consists of the consolidated conceptual model, the attachment of concepts to methods and the references from modeled objects to the structure of exchangeable data in the SQL-databases. Furthermore, it manages the process knowledge, i. e. communication and order processing, documentation of orders, order scheduling, etc.

Configured around the conceptual model, the trader will contain all knowledge of possible communication between the KBS'. Queries can be carried out in two different ways:

1. If the asking subsystem has carried out its query successfully before, it can communicate with the correct answering subsystem directly.
2. If the asking system neither knows where to get some information nor how it is structured, it can ask the trader by presenting a requested answer format, using the concepts of the conceptual model. The trader will look for methods and exchange data structures connected to the concepts asked for using algorithms that explore the conceptual model and find the related DB-schemata. It will then generate an adequate query which can be sent by the asking system if the query involves only one answering system. If a distributed query is necessary, the trader itself will ask the methods and join the answers.

The trader should not only support automatic querying but also the direct communication of the human agents behind the methods. They can ask the trader for information directly, using special communication protocols simlar to SQL. This feature is needed because of the group-oriented way quality management is realized today.

Both the modeling and realization of information exchange and presentation is future work of the WibQuS research group. But the proposed architecture of the information system indicates the way a repository can serve as a major part of a distributed quality information system and therefore can be used in distributed method modeling and specification but also in method support.

CONCLUSION

Starting from a description of a deductive and object-oriented framework for repository management, we have analyzed its usefulness in distributed modeling of quality management methods. While the general experiences resulting from the WibQuS group can be considered as promising, a number of problems concerning the organization of the modeling process and the quality of the technical support have been identified.

We have described how the spatial and conceptual distances in a distributed modeling process can be bridged by a repository embedded in a client-server architecture. We also indicated how a deductive and object-oriented DBMS like ConceptBase can be used as model integrator and communication support during the modeling process.

We pointed out that repositories are not only useful tools for system development but also for method support. The proposed architecture of the quality management information system based on the developed conceptual basis is only one step in this direction. Depending on the structure of the repository, there are a lot of other possibilities, even in the WibQuS research group.

ACKNOWLEDGMENTS

We would like to thank the members of the WibQuS research group for their interested participation in the time-consuming interviews. Furthermore we thank the members of the Lehrstuhl für Informatik V of the RWTH Aachen and especially M. Jeusfeld for a lot of helpful comments during the analysis of the WibQuS modeling process.

REFERENCES

Feigenbaum A.V., 1991, *Total Quality Control*; McGraw-Hill

Habermann, H.-J., Leymann, F., 1993, *Repository* (in German), Oldenbourg Verlag, Munic

IRDS Rapporteur Group, 1993, "Information Resource Dictionary System (IRDS): Service Interface Extensions", Technical report, ISO/IEC JTC1/SC21/WG3 N1566

Jarke, M. (ed.), 1993, "Database application engineering with DAIDA", Springer-Verlag

Jarke, M., Pohl, K., 1992, "Information Systems Quality and Quality Information Systems", *Proc. IFIP WG 8.2, Conf. Impact of Computer Supported Techniques on Information Systems Development*, Minneapolis, Minn.

Jarke, M., Eherer, S., Gallersdörfer, R., Jeusfeld, M.A., Staudt, M., 1993, "ConceptBase – A deductive Object Base Manager" (submitted), *VLDB Journal, Special Issue on Deductive Database Systems*, 1993

Jarke, M., Jeusfeld, M., Szczurko, P., 1993, "Three Aspects of Intelligent Cooperation in the Quality Life Cycle", Intl. Journal on Intelligent and Cooperative Information Systems Vol. 2, No. 4, Dec. 1993

Jeusfeld, M.A., 1992, "Update Control in Deductive Object Bases" (in German), Infix-Verlag, St. Augustin, Germany

Kosanke, K., Vernadat, F., 1992, "CIM-OSA: A Reference Architecture for CIM", *Proceedings of the IFIP TC5/WG5.3, Human Aspects in Computer Integrated Manufacturing*, Tokyo, Japan, 24–26 June 1992

Mylopoulos, J., Borgida, A., Jarke, M., Koubarakis, M., 1990, "Telos: a Language for Representing Knowledge about Information Systems", *ACM Transactions on Information Systems* 8(4), pp. 325–362

Pfeifer, T., 1993, "Management of Quality" (in German), Carl Hanser Verlag, Munich

Rose, T., Jarke, M., Mylopoulos, J., 1992, "Organizing Software Repositories, Modeling Requirements and Implementation Experiences", Technical Report DKBS-TR-92–2, University of Toronto

Scherr, A.-*W., 1988, "CIM — Computer Integrated Manufacturing (Somputer Steered Industry)*", Springer Verlag

Smolander, K., Lyytinen, K., Tahvainen, V.-P., Martiin, P., 1991, "Meta-Edit — A Flexible Graphical Environment for Methodology Modeling", *Proc of CAiSE '91*, Trondheim, Norway, Lecture Notes in Computer Science 498, Springer Verlag, pp. 168–193

Staudt, M., Nissen, H., Jeusfeld, M., "Query by rule, class and concept", *Special Issue on Knowledge Base Management Systems*, Applied Intelligence, 10/93

Wenzel, B.G.; 1991, *Information Modeling using EXPRESS*, Digital International GmbH, Munich