A Modelling Approach to the Realisation of Modular Information Spaces

Moira C. Norrie and Alexios Palinginis

Department of Computer Science
ETH Zurich, CH-8092 Zurich, Switzerland
{norrie,palinginis}@inf.ethz.ch

Abstract. We present a metamodel which forms the basis for the design and implementation of modular information systems supporting a cooperative working environment. The model consists of four separate, but interconnected, sub-models dealing with all aspects of modular systems from the database *meta and object model* down to a possible *storage model*. The database *connectivity metamodel* is crucial in supporting the implementation of the database and connectivity models which enable users to dynamically dock on a foreign database module. At the centre is the *user model* which serves to tie the other sub-models together and this reflects our human-centric approach to cooperative environments. Consistency of each database module is maintained through our model of personal and shared workspaces. The global consistency of interconnected database modules can be achieved through synchronisation and cooperation of the conflicting parties over personal and history data.

1 Introduction

When designing a complete information system, many invariants such as a clear information model, data consistency, efficient query and recovery operations, must be ensured in the lifetime of the application. On the other hand, free collaboration between multiple users, physical storage distribution, replication and interchange mechanisms must be enabled, which may often introduce undesirable side-effects to the information system invariants.

With the goal of solving the problem of legacy database integration, we have seen major efforts within the information system community in the areas of schema integration, data heterogeneity and federated databases [HM85, SL90, Bro92]. In the mean time, modularity has become a standard tool for software engineering and componentware is gaining broad acceptance in the object-oriented programming world. Nevertheless, every day new database applications are created still based on old information and distribution models. This results in a large impedance mismatch between data and application modelling and design techniques, thereby preventing the full exploitation of new technological advances, while retaining many of the old pitfalls. We notice that the current middleware emphasis is justified by its success to close the gap between such technological mismatches. However, we still believe that, for new applications built from

A. Banks Pidduck et al. (Eds.): CAISE 2002, LNCS 2348, pp. 245–261, 2002.

scratch, new models and techniques must be used, not only at the application level, but also at that of the information itself.

The overall complexity of distribution in an information system is a major drawback in terms of, not only its design, but also its implementation and efficiency. In the past years, many communication and synchronisation protocols [Gri98] that support distribution have been studied and proposed. Nevertheless most of them are at a very low level: They support object-orientation only at the level of programming languages and processes and not in terms of information abstractions and cooperative work activities. Thus, while they are useful to solve the problems of communication between distributed objects, a lot more is required to achieve the goal of modular information systems. The effort involved in designing and implementing a complete information system application is still a very time-consuming and difficult task.

We are therefore seeking both a methodology and a model to support the design and implementation of an information system in terms of modular components. To achieve this goal, the approach taken must be total and at a higher level of abstraction than just that of a communication protocol: It must aim to solve the problem of the distribution of information, rather than the problems of the communication and synchronisation of data arising from it. Existing distributed protocols can be integrated into the methodology and model by defining the operational specifications of the special partial problems that they address.

Our system focuses on the extensibility and reusability of databases in a user-centric approach, rather than purely on the underlying data. Information is indeed formed out of data and their consistency is crucial. Still, the perception of information is user and situation dependent. Each user must have a consistent information space, but it is not guaranteed that this will be part of a consistent extended space when participating in a community. Nevertheless, it is desirable for a user to dynamically participate in other communities. An information system must then be able to dynamically dock on other systems and resolve possible conflicts through cooperation and contract definitions.

Thus we are working on a logical rather than physical distribution schema. Contrary to a transparent distribution environment, the users view the entire information space in terms of logical, user-aware sub-spaces. As stated above, physical distribution techniques can still be used and applied to each logical application entity, but, by defining logical application modules and their interaction and evolution, we achieve a system suited to cooperative working environments.

We identify the following four major conceptual areas, with their respective models, that influence the design and implementation of such a system:

- A rich information model.
- Database modules and their interconnection model.
- A flexible storage model that supports cooperative working environments.
- A user model for our human centric approach and security aspects.

In the following sections, we will describe each of these models in turn and investigate the dependencies between them. We begin with a discussion of the

general requirements in Section 2. In Sections 3 to 5, we describe each of the sub-models, focusing on the features that enable better understanding and control of the distribution aspects of a modular information space. In Section 6, we examine how these models are combined. We give concluding remarks in Section 7.

2 Requirements

In this section, we briefly introduce the four sub-models proposed by examining the general requirements. An overview of the four components is given in Figure 1.

First of all, the information system must be defined clearly, completely and orthogonally in terms of an *information model* and its corresponding metamodel. This model must be kept as *simple* as possible to avoid unnecessary complications when distribution comes into play. "Simple" refers to the concepts used and the operations applied to the abstractions offered by the model. The model should be complete and orthogonal with respect to operations and constructs, thus ensuring the robustness of the system in delicate situations that may be introduced by distributed operation (disconnected operation, replication, partitioning etc.). The entities of the system must be uniquely identified, not only in the scope of the database module in which they reside, but also in the entire potential cooperation universe in which they might participate now or in the future.

We see a great significance in the existence of a *database connectivity model*, which models how the different peers should be connected and what kind of entities are represented alone or in conjunction with each other. We handle a database (or a piece of it) as an object itself which can have attributes, operations and can reference other databases, thereby building up hierarchies on which inheritance and encapsulation can be applied.

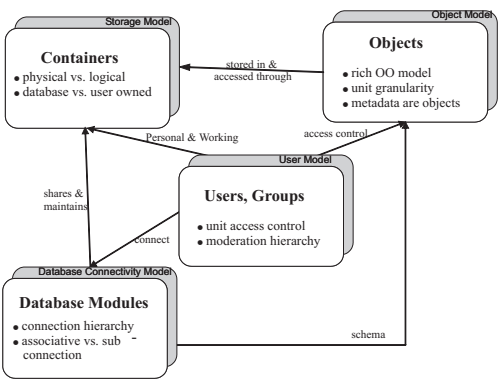

Fig. 1. Metamodel components

The distribution aspect is indeed tightly coupled with the storage of the information, making the introduction of a general object *storage model* inevitable. We use a model based on storage containers, which represent units of storage at a logical or physical abstraction level. A user view is an information space which is constructed dynamically from a set of storage containers in terms of set operators over these containers.

Last, but not least, a *user model* is required. As a result of our aim to support cooperative human environments, the user model is actually the interface and interconnection between all of the sub-models mentioned above. The user is aware of the high-level logical distribution and database connection models. Behind the scenes, the user model drives part of the storage model and gives admission to the visibility of the object model.

Before discussing each model in detail, we present an example to illustrate our vision of a modular information space. Assume we want to model a population database for Switzerland which tracks all inhabitants in databases distributed across the country and maintained by the cantons (autonomous regions of the Swiss confederation). The canton of Zurich may have a database where information on all inhabitants of Zurich are stored under the guidance provided by the federal inhabitants schema, located in the Swiss capital, Bern. Due to its autonomy, Zurich could enrich the basic inhabitants data, based on the schema from Bern, with other information and operations, such as information on local taxes/statistics and/or some local rules or restrictions. This schema extension would involve the creation of one or more local subtypes, e.g. `ZH_person` as a subtype of `person` as shown in Figure 2.

ETH Zurich, one of the two Swiss federal universities, is located in Zurich. Assume a restriction that employees of ETH Zurich must live in the local canton. ETH has access to the information about its employees from the local canton

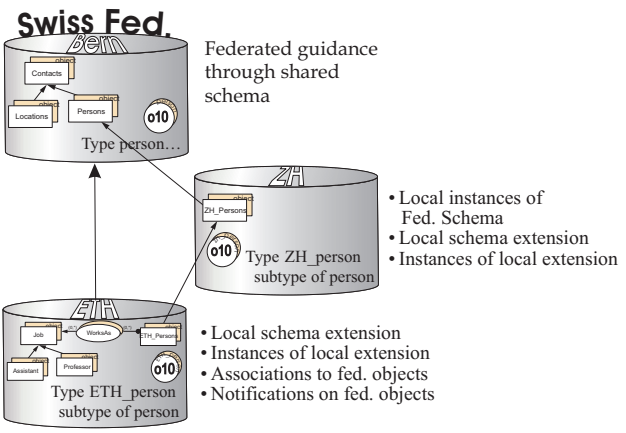

Fig. 2. Inhabitants DB Example

inhabitants database and, again, it is free to extend it through additional subtypes as required, e.g. `ETH_person` as a subtype of `ZH_person`. However, in this case, control of the inhabitant objects remains with the canton and ETH stores only the information units corresponding to its local extension in its database and not the whole object. Thus, only the methods and attributes defined for type `ETH_person` and not the inherited properties are stored at ETH.

We clearly see that the physical distribution is based on and driven by the logical application model. Such a structure could be further extended to deal with, for example, ETH internal organisations which may have requirements to share both schema and data, with or without taking control over the management of the relevant objects. At each level of sharing, it is, of course, necessary to specify which information is accessible to other systems. Thus both data and metadata are subject to access controls.

In addition, the situation may arise where a user wants to update data that is not within their control. For example, assume that ETH wants to change the employment conditions of one of its employees. ETH may update this information in the local database, but it ultimately requires the approval of the canton of Zurich, which issues work permits and processes tax information. In such a case, ETH may either be given full privileges to alter the information on the database of Zurich, or, it may apply for an update through a "change request" submission. Thus, the process is not simply one of update synchronisation, but rather of cooperation between the users of data.

3 Information Model and Metamodel

An information system is defined through its schema and the corresponding metadata defines the general concepts of the application. The metadata provides the system with crucial information, even at run-time, about the structure and status of the application data. This information can also be used to support the distribution strategy. Our aim is to give a user the ability to dynamically dock on a foreign database, use it and even extend it, without any administrator or designer intervention. It is therefore of crucial importance that the metadata are also represented and handled as normal objects.

Object-oriented metamodels [ABD+89] are powerful due to the fact that the notion of an object is a simple orthogonal abstraction, based on which, all of the concepts of an application can be modelled. Providing distribution of the object abstraction, in the form of types, entities or operations, will yield some kind of application distribution. The same applies to relational metamodels which have also been used successfully as a basis for distribution. However, as stated in the introduction, our overall aim is to obtain a methodology and a model to support overall information system application design. Therefore, the existence and completeness of an information model alone is not sufficient.

In our aim to support extensibility and reusability, concepts such as fine-grained object decomposition, collections of relevant objects and object relationships through references are important characteristics. We therefore need

an information model that incorporates such constructs. Through a rich object-oriented model that encapsulates many application concepts, the implementation of a distributed information system is greatly simplified. Although the addition of certain semantic constructs to the core of the system may initially appear to introduce unnecessary complexity, it in fact pays off in terms of simplifying application development.

We have based our prototype on the OMS Pro data management system [Wue00, KNW98]. It, in turn, is based on OM, an object-oriented model with support for object collections and associations, a rich collection algebra and object evolution [Nor93]. OMS Pro handles all information, metadata and data with the same object concept. In fact, when it comes to the implementation, metadata is handled slightly differently in order to achieve efficiency, but the logical behaviour is common to all database objects. As a result, when a client database connects to a server database, the client can have full access (subject to access controls) to the metadata enabling the remote data to have known type and structure. Moreover, the choice of a model which clearly separates notions of entity representation and classification within its type and value system [Nor95] allows us to even work with parts of an object or even to reference objects that are private. The system and model were both developed within our research group which gave us the freedom and resources to introduce distribution concepts into the core of the system. Due to the fact that OM is orthogonal with respect to operations and constructs, it demonstrates the benefits of making as many information systems concepts distribution-aware as possible.

Returning to our Swiss inhabitants example, the shared schema would be stored in a shared database in Bern. A canton could create `person` objects locally according to this shared schema. No special handling of this metadata must be undertaken due to the fact that distribution, partitioning, replication and synchronisation of metadata are treated as normal objects. The canton of Zurich can then extend the remote shared schema by creating locally additional metadata. For example, since local cantons manage taxes autonomously, the canton of Zurich could create a local subtype of the shared schema's `person` type with the relevant tax-related attributes and methods.

In Figure 3, we present the object and metamodel of the OMS database in the OM notation. Rectangles represent collections of objects and ovals represent associations between collections, i.e. a collection of reference pairs. Classification hierarchies are formed through subcollection relationships between collections as specified by `cisa` objects, indicated graphically through directed edges between collections.

In the shaded part of collection rectangles, the type of the member objects is indicated. Type hierarchies are built from subtype relationships represented as `tisa` objects. Active objects represent the operational part of the database through methods, triggers and macros (database operations).

One can see that metadata are also normal objects with an appropriate type and possibly methods. Of course, in the definition of the basic persistent type

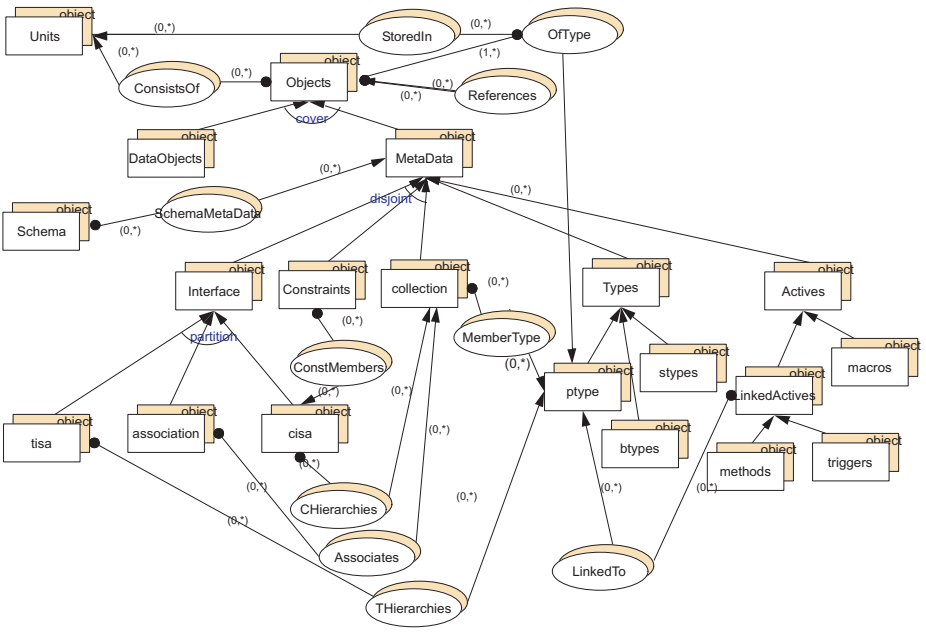

Fig. 3. Information Model and Metamodel

and object (`ptype` and `object`), we arrive at a classic fix-point situation where those objects must be defined using themselves.

Although we cannot provide details of the OM model and its metamodel here, the main constructs of the model can be summarised as:

- Unique objects which may be instances of arbitrary types with attributes of base/structured or reference type.
- Types and subtype (tisa) relationships.
- Methods, triggers and macros.
- Collections with subcollection (cisa) relationships and constraints such as disjoint and partition over them.
- Associations between objects participating in collections.

These are the logical constructs that actually influence database consistency and, as seen later, form one dimension that influences the distribution strategy. Although the presence of collections, associations and their constraints increases the complexity of the core system in comparison to a simple object-type model, we prefer to deal with these concepts once in the core of the system rather than having to provide ad-hoc distribution of rich constraints within the application.

The various forms of relationship metadata objects — `association`, `cisa` and `tisa` — are treated as special cases in our model. They represent the linking between objects used as the basis for connection and extension when interconnecting databases. After explaining the database connectivity model in the next

section, the dependency between these metadata objects and the database model should be clear.

Apart from the metamodel, we are interested in the granularity of information distribution. One can decide to allow distribution of object collections or choose to distribute at the granularity level of attributes. In earlier work [NPW98], we successfully used the granularity of an object unit. This is the set of attribute values of an object associated with a particular type, without its inherited attributes. For example, the object `o10` of type `employee` which is a subtype of `person` would consist of two separate units, namely that of `(o10,employee)` and `(o10,person)`. The abstraction of a unit as a basis for distribution is appropriate for the following reasons:

- `(ObjID,Type)` pairs can be seen as unique keys of searchable information.
- The system offers extension at a finer level of granularity than that of an object, providing flexibility without reaching the inefficient fragmentation of attribute level distribution.
- We provide multiple inheritance and context-aware views of objects by dynamically composing object instances from units.
- The user access privileges can be applied to the unit level, providing fine-grained security.

Last but not least, dealing with units at the physical level of the system, enables us to provide rich functionality and flexibility to the upper logical abstraction levels, while keeping the underlying implementation simple and efficient.

4 Database Connectivity

We now discuss some database connectivity issues and the corresponding part of the metamodel. As stated previously, we want to create a dynamic, shareable, distributed information system. The usual client/server architecture would be very limited for this purpose, as it should be possible that a database shares its data and schema and, at the same time, acts as a client on other databases. A database module should be autonomous, allow other modules to extend it and be able to be an extension from other modules. However, allowing such flexibility could introduce some problems. What if, for instance, a cycle of interconnected databases occurs? Where must the data, and above all the metadata, be stored? How can we control propagations among database modules?

To give solutions to such problems, one must define what a database module is and how it relates to the information model. As seen in the previous section, a database is defined by its schema. The question that arises in a distributed interconnected environment of database modules, is how can we decide which part of the schema belongs to which module and, further, when can we say that a database module is consistent?

The straightforward answer used in many object-oriented systems is to use the transitive closure of objects, defined as the information of an object plus the information of all objects that are linked or referenced from it. If we have all this

information and it is consistent, then the module itself is in a consistent state. Recall now from our requirements that a user of a database module should be able to dynamically attach to and extend a foreign module. If we base our model on the transitive closure concept, such interconnected and linked modules could never be disconnected again! We therefore looked for another approach.

We distinguish two kinds of module connections:

- Applications that are logically autonomous and one can shift the focus to another logical application through linked information.
- Applications that are more tightly linked together and build a hierarchy.

It is clear that, in the first case, the consistency of the application is separate from the consistency of the linked information. The user is focused on the application domain of the module but has the opportunity to attach dynamically to further information and use another application. The user can later detach from the other module without influencing its consistency. If desired, the first working scenario can migrate to the second, more strict scenario, where the two parties are willing to cooperate and build a community that must also be consistent due to some kind of contract (new extended consistency).

We therefore see a database module as a logical application abstraction. When modelling large applications, one can easily come up with sub-schemas interconnected with each other via association, subtype or subcollection relationships, representing logical domains of the whole application. Even graphically, we tend to draw partial application spaces adjacent to each other. Based on these considerations, we introduce the following definition:

> "A consistent database module schema consists of all metadata objects of a logical application domain and all further objects that build up their transitive *application consistent* closure."

Notice that the transitive *application consistent* closure is a subset of the absolute transitive closure. Taking advantage of the reference semantics of an association and, using the unit model described in the previous section, we are able to define the application borders through those constructs.

In OM, associations are the connection points between objects. If we were to ignore them, we would reduce the logical application, but could then easily ensure the consistency of the remaining parts of the application domain. From a particular part of a schema, we can regard associations as the bridges to the neighbouring database modules. If we close these bridges, we can still remain and work effectively within our own information island, as long as that island is itself a meaningful sub-space of the overall application information space.

Some contexts in which an application object resides can also be set apart if desired. As mentioned before, OM models object roles that can define context-sensitive properties and behaviour. For example, a person can have both a student and a private context for two different application sub-spaces. If it makes logical sense, a context can be omitted. Thus, the specialisation relationships declared through subtype (tisa) and subcollection (cisa) relationships can also

be seen as bridges to neighbouring contexts. Because the object model is physically structured in terms of units, the partitioning of objects over contexts is straightforward. For this reason, we see in Figure 3 that associations, tisas and cisas are all regarded as special forms of interface metadata.

In summary, a database module represents a logical information space, defined by a consistent schema and composed of objects according to that schema. Through connectivity with other database modules, the information space can be extended through associations and/or contexts defined by interface metadata. The database modules can then build-up a hierarchy. A module can view only the immediate connected modules, thereby avoiding major propagation and security issues. If a module A wishes to use a module B that extends another module C, then the module A must connect explicitly to module C. The system supports this by providing the dependencies of a module to the connected clients.

The resulting connectivity model is shown in Figure 4. We further distinguish connections (`Connection`), based on the kind of interface metadata used to connect the databases, into Associative (`AssocDB`) and Submodule (`SubDB`) connections. This is mainly due to the fact that the connection through an associative bridge, represents a more loosely-coupled connection than that of a submodule connection. Making this distinction enables us to optimise aspects of performance in the implementation.

In our inhabitants example, every canton would have at least one database module. A shared database module, namely that of the federation, is maintained in Bern. The federation would have many such modules for further applications, such as federation laws and foreign policy. Associations across these logically separated application modules enables them to be connected to each other. Suppose the traffic department of the canton of Zurich now wants to work on traffic data that they created locally and associated to the inhabitant data based on the shared schema. They can perform this operation by simply using their traffic database module and connecting to the inhabitant module of Bern. There is no need to connect to the federation foreign policy module, even if this is interconnected to the inhabitants module. An associative connection between the database modules is created automatically.

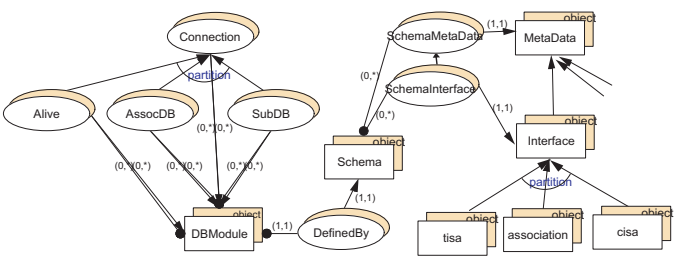

Fig. 4. Connectivity Model

On the other hand, ETH also uses the inhabitants module, this time of the canton of Zurich, and can extend the schema by introducing new subtypes locally, such as `ETH_person`, `student` and `professor` with new attributes and methods. The information units of each student or professor object will be stored locally, while the person units will still be maintained in the canton of Zurich database. Once ETH has established such a connection, it has a database module that is strongly connected to the inhabitant database of canton of Zurich. In effect, they are building a submodule hierarchy. The information units of each student and professor must be available to ETH from the different cantons. Moreover, due to the fact that we actually distribute objects based on logical criteria, the system can automate the replication process using the metadata [Pit96] and copy the student/professor inhabitant units locally to ETH, increasing not only the response time, but also the ability to work further in a case of a connection failure.

Having this connectivity model, we still have not answered the question as to where the objects are stored and how the users are involved. To answer this, we first need to present a storage model appropriate for a distributed, cooperative working environment.

5 Storage Model

A major property of information systems is to guarantee consistency of the stored data. Most of the proposed approaches are based on a short transaction scheme. An update, typically carried out from a task or a batch command, locks a resource, applies the update and frees the resource again. In our system which focuses on cooperative working environments, we take another approach.

An OMS transaction represents a particular logical work activity on the database which may even span several OMS user sessions. The length of such a transaction will definitely be too large to apply traditional locking techniques. Apart from the logical consistency of the system, we must ensure further physical consistency. All user interactions must be made persistent automatically and recovery must be ensured in the case of failure. OMS Pro was originally designed as a rapid prototyping tool, and, in such a development environment, physical consistency is crucial since "mistakes" are not excluded. Further, a human cooperation environment could lead to inconsistencies due to the different perspectives of each user.

We therefore built a system that gives each individual the flexibility to always secure his/her work and to support the resolution of logical conflicts by either providing synchronisation tools or a personal view over the information. The decision to use information units as our level of granularity, as discussed in Section 3, supports such a strategy by providing quite small granularity and thereby restricting the level of conflicting information. In this section, we focus therefore on information units as the physical level of information storage, rather than whole objects.

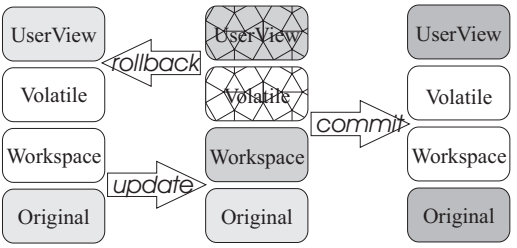

Fig. 5. Information Spaces

The information units are stored in what we call "information storage containers". These containers may be either physically stored or virtual containers derived from other containers (for example, views defined in terms of set operators). This means that to retrieve a unit given its key (recall from section 3 that the pair (ObjID,type) is the unique key of a unit) from a virtual container V, the request sent to V will be propagated to the containers in terms of which V is defined. This may seem complicated initially, but it allows us to be very flexible and enables us to model many distribution scenarios. With special storage structures, we are even able to reduce the overheads introduced.

To give a better understanding of how we work with information containers, let us study the use of such containers to implement the single user storage solution of Figure 5. It comprises one logical and three physical containers. *Original* is a physical container holding the database consistent state. This container is implemented in a similar manner to a conventional object repository, providing indexing, recovery etc. Each user has his own *Workspace* container where all the changes are immediately made persistent. A workspace is also a physical container but, because it holds a small number of units, indexing may not be so crucial. Temporary objects are stored in the *Volatile* container which, although it is a physical container, is implemented in memory. The *UserView* container is then a virtual space that is defined as follows:

$$UserView = \{x |\ (x \in Original \wedge x \notin Workspace)\ \vee \\ x \in Workspace \vee x \in Volatile\}, \quad \text{where x is a unit}$$

The user always views the UserView container, while the system writes updates only to the Workspace or Volatile container, thereby avoiding fragmentation of the Original container and decreasing the possibility of loss of information due to failures in the Original container. On commit, the UserView is checked for consistency and, if the check succeeds, the changes in Workspace are "melted" into the Original; the Workspace and Volatile containers are emptied for the next session. To support some functionality required for logging, versioning and rollback through the different logical commit stages, the Workspace is actually copied to a history container. Any snapshot of the Original container over time

can consequently be reconstructed from the subsequent union of its History containers.

The abstraction of information containers, combined with the concepts of information unit uniqueness and small granularity, gives us the flexibility to, not only easily introduce new virtual storage containers by just defining a name and the corresponding dependencies between others, but also to optimise the implementation of physical containers or even re-distribute the units of one physical container into many other containers. For example, the Original container in OMS is actually partitioned into three physical containers according to the category of units in terms of whether they are meta-, user- or classification-data.

To extend the single user case just discussed into a multi-user cooperative environment, we next present the user model.

6 User Model and the Model in its Entirety

The user has an important role in our system in actually binding together the models discussed so far and their properties (see Figure 1 or Figure 6 for a more detailed view). Therefore, while presenting the user model, we will also explain the model in its entirety.

One can notice that the user model authenticates access to objects and forms connections between database modules. Although, at this stage of our project, security is not a major focus, we model and implement access privileges through user groups. We perform security checking on the physical information level, i.e. that of the information unit. Introducing security at such a fine-grained level causes some additional loss of efficiency of the system. For the moment, we have a naive and rather slow implementation that we aim to improve in the future.

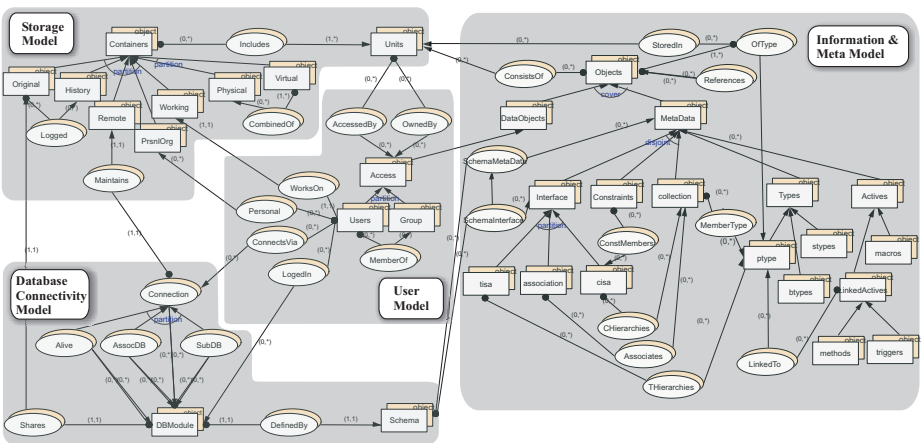

Fig. 6. The total detailed Model

It is worth noting that the User notion is also an object of our information model, but it has to be treated specially at the physical storage level to increase performance.

Having introduced information containers as storage abstractions and explaining the single user solution, let us now extend the abstraction in the context of cooperative working environments. Every database module has a single Original space where it stores consistent units defined by the database schema as mentioned in the previous section. When a user logs into the database and at the same time connects to a remote database module, they will view a union of both modules (subject to access controls). The uniqueness of the objects in the new environment is guaranteed by a hierarchical registration process which was presented in a previous publication [NPW98]. The user can then decide to use the connected database either by extending it through the creation of tisa relationships, or by designing affinities through the creation of associations. As mentioned before, all changes will immediately become persistent in the user's working containers (*Workspace* and *Volatile*).

For every connection, the system maintains a virtual container *Remote* (see upper-left corner of Figure 6). It comprises all objects made accessible through the connection. The user can further extend its functionality by introducing a physical sub-container of *Remote* that acts as network cache. The system can be directed to keep in the network cache all referenced objects that are requested together with their consistent transient closures. For large amounts of data, this could be time consuming, but it enables additional important features such as disconnected operation.

When the user decides that they have reached a logical working milestone, they would check the new state and, if consistent, the system will "melt" the changes into the shared Original container. This working scheme can be observed frequently in development processes such as web site development and Software Configuration Management [Est01]. Developers will change some information from the current consistent state, test them for mistakes and inconsistencies and, when satisfied with these changes, the changes are deployed. This is exactly how our system core works. Using our system, it is therefore possible to implement such an application with minimal development effort, by exploiting the rich functionality of the cooperative working schema.

Let us now discuss the possibility of two users changing a shared object simultaneously. As mentioned before, we usually will not lock the objects. Thus, the possibility exists that the first user changes an object field, while another user is also working on it. But recall that both users will read the shared object and apply any change to their own Workspace. When one of the users first commits objects in the shared database, the other could then find that his database view is in an inconsistent state. Here we refer to the extended consistency of the information space formed by the connection of the database modules between each user and the shared module. The local consistency will stay intact due to the absence of the bridge objects that extend the information space, as explained in section 4. But what motivates us to allow such a possibility?

In a cooperative working environment, and most of all when the parties are loosely connected, an information unit can evolve in an application domain without the awareness of foreign application domains. In such cases, we arrive at a conflict situation. The involved parties can view the different versions and cooperate to resolve the conflict. In our system we support two different scenarios. On one hand, the cooperative environment may have some kind of hierarchical structure, whether it is an organisational, operational or even task-oriented hierarchy. Thus giving priority to the changes introduced by the party higher in the hierarchy. On the other hand, an application could support a more liberal, democratic way where multiple perceptions can be accepted. We therefore model this fact by associating the user model with the storage model as shown in Figure 6.

In both cases, the server database should only be updated if the docked user has update privileges. If not, the user will get a personal view of the remote database for any changes that will be made: The changes will be melted into the user's *Personal* container ensuring the consistency of the user information space and avoiding the introduction of further conflicts with other users of the shared module, thereby giving a solution to the democratic view-point with personal versions.

To support the hierarchical working model, the users of a particular database are building up a priority hierarchy. This may enable more users to access and even update data, while giving moderation privileges over the data and its synchronisation in conflict situations. This scheme is ideal for multi-user applications working in a workflow manner commonly used in all hierarchical organisational working models. In the case that the working model changes over time, it is always possible to again publish your personal view of the information or vice-versa.

Imagine in our ETH example that a teaching assistant wants to reduce their working time by half. The personnel department makes the appropriate ETH internal changes to the database. Recall that the canton of Zurich tracks all working profiles to adapt the taxes or work permits of its inhabitants. Consequently, the occupation information must also be altered in the database of Zurich. Because ETH has no direct write access to the data, the changes are stored at the ETH personnel container and a notification of the situation is sent to the canton of Zurich database. The responsible person in the employment centre of Zurich, the *moderator* that handles ETH cases, is informed from the system about the request for occupation change. He can either accept the change by adopting the ETH personnel copy of the employment object or resolve the inconsistency due to this change in cooperation with the ETH personnel department. In this way, the central office retains control of the data, but clients of that data may notify them of changes.

7 Conclusions

We have identified four major concept areas that must be investigated when building a distributed information system for a cooperative working environment. The model presented defines the concepts of each area and interconnects them to support both data and schema sharing in modular information spaces.

The process of modelling our system in this way, led to a better understanding of the existing concepts that influence overall system design and the interoperability between them. This, in turn, led to improved functionality and efficiency of the current platform over an initial prototype.

To perform such metamodelling, it is important to use a rich and expressive data model that increases the semantic information about the data, which can, in turn, be used to drive the distribution. Moreover the overall implementation of the application will be easier by using distribution, replication and synchronisation concepts already introduced in the core of the system with a clear logical model. A flexible object storage model must exist to support distribution and interoperability between users. The granularity of information is crucial, not only for reasons of efficiency, but also flexibility. Finally, the user plays a central role in the system, controlling the connectivity within his environment. Two working strategies are supported by the system: A user privilege schema organised hierarchically, that supports user cooperation and a more autonomous solution based on versioning.

References

[ABD+89] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Meier, and S. Zdonik. The Object-Oriented Database System Manifesto. In *Proc. 1st Int. Conference on Deductive and Obect-Oriented Databases (DOOD)*, Japan, 1989. 249

[Bro92] M. L. Brodie. The Promise of Distributed Computing and Challenges of Legacy Information Systems. In D. K. Hsiao, E. J. Neuhold, and R. Sacks-Davis, editors, *Proceedings of IFIP DS-5 Semantics of Interoperable Database Systems*, volume 1, pages 1–30, Lorne, Victoria, Australia, November 1992. 245

[Est01] J. Estublier. Objects Control for Software Configuration Management. In *Proc. of 13th Conference on Advanced Information Systems Engineering (CAiSE'01)*, jun 2001. 258

[Gri98] Rebecca E. Grinter. Recomposition: Putting it all back together again. In *Computer Supported Cooperative Work*, pages 393–402, 1998. 246

[HM85] D. Heimbigner and D. McLeod. A Federated Architecture for Information Management. *ACM Transactions on Office Information Systems*, 3(3):253–278, 1985. 245

[KNW98] A. Kobler, M. C. Norrie, and A. Würgler. OMS Approach to Database Development through Rapid Prototyping. In *Proc. 8th Workshop on Information Technologies and Systems (WITS'98)*, Helsinki, Finland, December 1998. 250

[Nor93] M. C. Norrie. An Extended Entity-Relationship Approach to Data Management in Object-Oriented Systems. In *12th Intl. Conf. on Entity-Relationship Approach*, pages 390–401, Dallas, Texas, December 1993. Springer-Verlag, LNCS 823. 250

[Nor95] M. C. Norrie. Distinguishing Typing and Classification in Object Data Models. In *Information Modelling and Knowledge Bases*, volume VI, chapter 25. IOS, 1995. (originally appeared in Proc. European-Japanese Seminar on Information and Knowledge Modelling, Stockholm, Sweden, June 1994). 250

[NPW98] M. C. Norrie, A. Palinginis, and A. Würgler. OMS Connect: Supporting Multidatabase and Mobile Working through Database Connectivity. In *Proc. Conference on Cooperative Information Systems*, New York, USA, August 1998. 252, 258

[Pit96] E. Pitoura. A Replication Schema to Support Weak Connectivity in Mobile Information Systems. In *Proc. of 7th Intl. Conf. on Database and Expert System Applications (DEXA)*, sep 1996. 255

[SL90] A. Sheth and J. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, September 1990. 245

[Wue00] A. Wuergler. *OMS Development Framework: Rapid Prototyping for Object-Oriented Databases*. PhD Thesis, Department of Computer Science, ETH, CH-8092 Zurich, Switzerland, 2000. 250