MODULARIZATION IN AN
OBJECT-ORIENTED KNOWLEDGE BASE SYSTEM

Gerhard Steinke
Universität Passau
Fakultät für Mathematik und Informatik
Postfach 2540
D-8390 Passau, Germany
Email: steinke@andorfer.fmi.uni-passau.de

Abstract

Modularization of a knowledge base system has the following goals: improving system response time, providing a more secure system and adding the capability to load or unload individual modules. But the object-oriented characteristics of a knowledge base system make modularization a complex task, due in large part to the many inter-relationships of the objects.

In this paper we suggest a modularization strategy for the knowledge representation language Telos, based on the structure of the language and with consideration for the security framework of the Group Security model.

1 Introduction

With such broad concepts in our title, it is necessary to define our understanding of these terms. Modularization is the process of dividing the basic entities of data in an information system into modules, based on some physical and/or logical criteria. Object-oriented and knowledge base are terms, which, although common today, are used to describe a wide spectrum of systems. Our understanding of an object-oriented knowledge base system is based on the knowledge representation language Telos [1], which integrates deductive rules and integrity constraints as well as an interval-based time calculus into a powerful, structurally object-oriented framework with full classification, generalization and aggregation hierarchies.

While an application is small, there is little incentive to consider a grouping of objects. But as the size of a knowledge base increases, it becomes clear that one should consider modularization. It is especially the multitude of relationships within an object-oriented knowledge base system, that make the specifications of modules a complex task.

2 Modularization Goals and Problems

The process of modularization creates clusters of information based on some physical and/or logical structure. The following subsection describes some goals of modularization. They include improving system response time, providing a more secure system, adding the capability to load or unload individual modules, and decreasing the size of the knowledge base. Problems in achieving the above goals are caused by the difficulty in finding a satisfactory structure to handle the inter-dependence and inter-relationships among the objects in the knowledge base. Such problems are described in section 2.2.

2.1 Goals

A request for information from the knowledge base must normally search through the entire knowledge base to find the answer(s). One goal of modularization is a reduction...
in the search space when looking for a particular fact. This reduction in the search space could be achieved if one could determine that the answer to a particular knowledge base request can only be found in a specific subset of the knowledge base, i.e., a limited number of modules. Then only these modules would have to be searched to find the response to the request. With a reduction in the search space one may expect faster response from the system.

Another goal of modularization is to divide the knowledge base into logical sections so that, depending on the application or task, only a limited number of modules, user and/or system specified, need to be present at a particular point in time. When accessing only a subset of the knowledge base, the response time can again be expected to improve.

Also, if the knowledge not needed is unavailable, i.e., certain modules are not even loaded, then the security risk of a user gaining access to information for which no access permission exists, is reduced—since a portion of such information is not even loaded. For example, if a user is only concerned with information on the medications available at a hospital, then knowledge concerning patients, doctors, and charts does not need to be present.

The existence of modules implies that they may be loaded or made available on an individual basis. If modules can be activated individually, then they can also be saved individually. By saving one means, made permanent, or no longer accessible, i.e., no longer online. If changes are made to a small portion of the knowledge base, it would not be necessary to "resave" the whole knowledge base, but only those portions which have been modified. If a user has finished work with a certain portion of the knowledge base, it could be released. Thereby it would be less likely for someone else to gain unauthorized access to that portion of the knowledge base.

A definition of modules, so that all objects which a task may access appear in a single module, would be ideal. Then a task could receive access permission to a module instead of a large number of individual objects.

Another objective of a module definition, is that one should be able to determine, with a minimum amount of effort, into which module a particular object belongs. Similarly, if one is looking for a particular object, one would like to have a good idea which module(s) needs to be searched, and which modules could not contain the desired information. This aspect of the definition of modules can be achieved, for example, if the module boundaries relate to the structure of the knowledge representation language. If one is looking for a particular type of object, say an attribute, then one would restrict the search to modules which contain attributes.

2.2 Problems

Most object-oriented systems have an aggregation of objects in terms of a class structure. But beyond the class structure, no larger object structure is common. In Telos, though, even the class structure is open, since objects within a class can just as well belong to, or be related to objects within other classes.

Some module specifications result in objects being assigned to several modules. An example is the worlds approach [3], which defines modules or object aggregations based on relationships to a particular concern or purpose. The clustering of objects is compared to different blueprints which are required in order to view the whole building. Just as each blueprint presents a different view of the whole, so each world presents a different aspect of the whole object base. The definition of these relations defining the different worlds is left to the user and the handling of elements common to different worlds is unclear.

The loading or making available of individual modules is a complex process. Whereas in a file or database system, one can open those files that are required, in an object-oriented system, the "opening" of a portion of the global object base is difficult, due in large part because of the interconnections and integrity requirements. Concepts dealing with portions of the global information system such as versions of files, and the deletion of files no longer required, are not found in object-oriented systems.

It is a complex problem to define modules which are user or task related. It would be advantageous if all objects which may be accessed by a particular user or task, are grouped together into a module. If all objects that task X may access are grouped into a single module, and the module contains no other information, then one would have a minimum search area and be able to retrieve information in the shortest amount of time. But this basis for modularization is more complex if the information available to one task overlaps with information permitted to another task. One could then consider having three modules: one module containing information only permitted to task X, another module containing information only permitted to task Y, and a third module consisting of information accessible to both task X and Y. As the number of tasks increases, the number of
modules would increase even faster. A change in access permission to an object, may result in changes to many modules or even the creation of new modules.

On the other hand, one could accept the fact that objects may appear in various modules. This duplication would require further processing when an object which exists in several modules is changed. Changes may also have to be made in modules which are currently not accessible, meaning that they have to be loaded. Thereby one defeats the purpose of making only the module for the current user available.

3 Overview of our Knowledge Base Environment

Our environment for creating modularization specifications is found in the knowledge representation language Telos, which is implemented in the knowledge base management system, ConceptBase [4]. The Group Security model provides access control for ConceptBase, and includes access control permission to complete modules. Section 3.1 provides an overview of Telos and section 3.2 describes the Group Security model.

3.1 Knowledge Representation Language Telos

Telos is a knowledge representation language which supports the concepts of object identity, instantiation, aggregation and specialization. Telos' structurally object-oriented framework generalizes earlier data models and knowledge representations, such as entity-relationship diagrams or semantic networks, and integrates them with temporal information and with predicative assertions. A formal description of Telos can be found in [1].

An example is used to illustrate the language [5]. Figure 1 gives a graphical representation of an example knowledge base in Telos. Figure 2 provides the textual format of the Telos syntax for the same example.

The object Person is declared as an instance of the system object INDIVIDUALCLASS, meaning that Person is both a class and an individual. The class Patient is defined as a specialization of Person. It has one attribute: takes. This attribute is itself an object that links Patient to other objects, in this case the class Drug. The object Drug has an attribute heals which is related to the class Symptom.
Predicative assertions are objects which are integrated in Telos as special attribute values of the metaclass Assertion. Their role as either deduction rules, integrity constraints or queries is defined by the way assertion objects, so called AssertionStrings, are attached to other objects by attribute links.

Telos maps all relationships of a complex object definition, including attributes, rules and constraints, to a single data structure of so-called propositions. The propositions corresponding to the example above are shown in figure 3.

```
Proposition(Person, Person, -, Person)
Proposition(Patient, Patient, -, Patient)
Proposition(#1, Patient, *isa, Person)
Proposition(Drug, Drug, -, Drug)
Proposition(#2, Patient, takes, Drug)
Proposition(#3, Fenta-forte, *instanceof, Drug)
Proposition(Jack, Jack, -, Jack)
Proposition(#4, Jack, *instanceof, Patient)
Proposition(#5, Jack, drug1, Fenta-forte)
Proposition(#6, #5, *instanceof, #2)
Proposition(#7, Drug, heals, Symptom)
```

Figure 3. Proposition form of Telos example.

So-called individual propositions with identical first, second and fourth component are drawn as nodes. The other objects are called attributes with two special cases: attributes with third component (label) *instanceof are called instantiation attributes and attributes with label *isa are called specialization attributes. For propositions of the form

```
Proposition(oid, x, *instanceof, c),
Proposition(oid, c, *isa, d)
```

x is called instance of c (c is the class of x), and c is called the subclass of d (d is a superclass of c).

Additional temporal components, present in the full Telos language, are left out of these 4-ary propositions for a more concise presentation. Our modularization effort focuses on a grouping these propositions.

A Telos knowledge base contains the objects Proposition, Individual, Attribute, InstanceOf and IsA which have as instances all propositions of the knowledge base with a form indicated by the class name. The structural properties of instantiation, aggregation and specialization are defined as built-in rules and constraints. Tokens are objects which are at the bottom level in that they are not be instantiated.

3.2 Group Security Model

The Group Security model provides a basis for organizing access control components in an object-oriented knowledge base system, in terms of real-world group structures based on the task concept.

The Group Security model is graphically represented in Figure 4 using Telos syntax. Thin links with no labels stand for instantiation relationships, thick grey links for specialization and the labeled links are attributes.

The top rectangle in figure 4 contains the five components of the basic model: user, role, task, access type and object. Users are assigned to a task by a role attribute link, which defines a user’s relationship to the task. Permission to access objects in the knowledge base is awarded to tasks. The access type link from a task and an object, determines what action a task is permitted, with respect to the particular object. A module is one type of object to which access permission may be granted.

The Group Security model is itself a part of the knowledge base. That means that objects of the Group Security model, similar to other objects in the knowledge base, are owned by tasks and can only be modified by users in those tasks that have the appropriate access permission.

The bottom rectangle in figure 4 presents the full Group Security model, including the relationships among the basic components. Further description of the Group Security model can be found in [2].
4 Proposed Module Definition

The following modularization concept is proposed, based on the knowledge representation language Telos, and its implementation in ConceptBase, along with the Group Security model. The module criteria, shown in figure 5, follows structural concepts found in the Telos language, is application oriented, divides the temporal aspects into historical and current, and considers the specifications of the security model.

All system objects are placed into one module, called the system module. This module would always be available. It would contain system and meta class objects.

A set of modules then exists for each application. All objects which define classes, class attributes and instances which are not tokens, for a particular application are placed into a module, called the application module. This module would include all rules and constraints, since rules and constraints are associated with class objects.

The system module and the application module provide the overall object framework for a particular application. While these two modules are essential, access to any object within the modules requires appropriate access permission.

The following three modules contain the actual data at the token level, i.e. the instances and attributes at the lowest level. If the application module resembles a data dictionary, in database terms, then the following modules would represent the contents of database records.
All individual objects, i.e. nodes which represent objects at the token level, are placed in a module called the individual module. This module is actually a list of identifiers of all token objects which occur within this application. This module is there to prevent duplicate identifiers.

Each group of instantiation links, between token objects and the class from which they are an instance of, are placed in separate modules called instance modules. A separate instance module would exist for each class found in the application module.

All instances of an attribute of each attribute object class also are placed into separate modules called attribute modules. They contain the attribute link objects and the instance links of that attribute. Attributes may themselves have attributes, which would also be included in this module.

The objects representing the Group Security model are placed into two types of modules. The attribute links from a task to the objects which that task may access, are placed into separate modules for each task, called the task modules. These modules are checked frequently, since during each access of the knowledge base, a check is performed to ensure that the current user is permitted access to the object. All other objects in the Group Security model are put into the security module. This would include the User and Task objects with instances and other attributes.

Current and Historical Modules
Modules are also defined on a temporal basis, current and historical. All objects which are no longer valid, i.e. those which have been deleted or have expired, are placed in historical modules. The historical knowledge base contains the same definition of modules as in the current knowledge base.

Figure 6. Module structure for a hospital knowledge base system.
Example
Figure 6 shows an example of the modularization of a hospital knowledge base system. The class objects with the attribute definitions, rules and constraints are placed in a module called Hospital.Application module. This would contain class objects such as Person, Employee, Patient, Drug and Symptom with attribute objects such as age, salary, takes and heals.

All identifiers of the tokens of all classes defined in the application module are found in the Hospital.Individual module. That means that the identifiers of the instances of the classes Person, Employee, Patient, Drug and Symptom are found in one module.

The instance links from the token objects to the class objects are found in instance modules, such as Employee.inst and Patient.inst. The attributes of each token are placed in modules by attribute type. The objects containing the ages of each token of the class Person are in one module called the Person.age.attr module. Another module called the Employee.salary.attr module would contain the salary objects of all Employees. Similarly, a separate module, Patient.takes.attr, exists for the takes attributes.

In addition, for each task a module exists which contains all the links from a task to the objects which the task may access, eg. the Charting.task module.

5 Evaluation and Implementation
The implementation of the modularization definition described above is taking place in ConceptBase, a knowledge base management system developed at the University of Passau.

This module definition relates closely to the structure of the knowledge representation language. That makes it rather straightforward to determine in which module a certain object belongs.

For requests which concern the whole population, or groups of the population, this module structure is good since access control permissions can frequently be assigned to a module. For example, if a user may see all the instances of a particular attribute, such as all the ages, or all the salaries, then access permission to a module can be given. If there are three groups of patients and they are handled separately, then three subclasses of patients could be defined, perhaps, medical, surgical and out-patient. The objects for each subclass would then be in different modules, so again access can be assigned to a complete module.

But the modularization scheme is not helpful for the situation where all information for one Patient is requested. Then several modules must be accessed, one for each attribute that Patient has. If such requests are common in an application, then another modularization concept should be chosen. ORION [6] is an object-oriented database system, in which the instances of a class are also placed in separate storage segments. But a user may specify a cluster message, to indicate that the instances of specified classes are to be placed in the same segment. While this presents additional complexities in terms of control, it would reduce the number of modules required.

This definition of modules also reduces the necessity of making available the full knowledge base. If a user is only concerned with determining the average age of patients, then the only attribute module required would be the age module. Salary and drug descriptions are not required to be available.

Another benefit of this module definition is that, within all modules associated with an application, there are no duplicate objects. The fact that Jack may be both a Patient and an Employee does not result in any duplicate objects. That is because the identifier Jack exists in the Individual module and the fact that he is a Patient and an Employee exists in two different instance modules. The Patient and Employee instance modules, and their attribute modules, may be loaded independently of one another.

Logically each object in the knowledge base requires a link to a particular module. In the Prolog storage system, of storing the propositions, this would double the number of objects in the knowledge base. But the relation of an object to a module is implemented by appending a module name to each object identifier, instead of adding module links to each object. Object Jack would have the identifier Jack.hospital.indiv. The module identifier would not be visible to a user.

The integrity checker in ConceptBase is described in [5]. But now, one is not only concerned for the integrity of the knowledge base as a whole, but also for the integrity...
of subsets of the knowledge base. The integrity of the knowledge base must be checked at various levels, be it for the subset of the knowledge base which a particular user may access, or be it for a number of modules to determine if they provide a "useable" knowledge base. Each user must have a knowledge base, which from his point of view and from the information that he can access, satisfies the accessible integrity constraints.

6 Further Research

Further research is necessary to improve the application and implementation of the suggested modularization structure.

Research is required to ease of the process of integrating overlapping applications. This involves determining what objects in the separate applications represent the same entities. For example, the class person may exist in several application modules. On the one hand, one wants the class definition to remain in each application module, so that each module could be loaded independently. But should several application modules be present at the same time, only one definition of person needs to be present, if indeed the definition of person is the same in each application.

On the other hand, classes in different applications may represent the same, but have different names, e.g. in one application they are called employees and in another application they are called workers. A particular individual, Bob, may exist in two applications--whereas the question needs to be answered, if they are indeed duplicates or whether they represent different persons who just happen to have the same identification.

When several applications are loaded at the same time, these duplicate objects must receive special attention. It should be possible during the loading of different applications to make or eliminate distinctions in names. The system must query the user concerning objects which may be duplicate, in order to know which objects represent the same information, and which objects it needs to modify in order to keep them distinct.

The handling of duplicate objects becomes more complex when one considers that some of the duplicates may not even be visible to this particular user. A user may add even more objects with the same identification, since he may not know that such objects already exist.

Further research is required to let the knowledge base help the user in knowing which modules need to be loaded together and which modules may not be loaded at the same time, in order to have a consistent knowledge base. For example, an attribute module may not be loaded if the corresponding application and individual module is not present.

One could also add a "door" to each module, which provides information concerning the contents of the module. For example, the information stating which tasks may access objects in the module, may be contained on the door. Then a task that has no access to the module, would not be able to search that module. Other information on the door to each module could be loading requirements and co-requisite modules.

One could also imagine that the module definition could be implemented as an extension to the Telos language. Further research is required to formalize this modularization concept.

Research is also taking place to ensure the integrity of the subset of the knowledge base which a user may access.

Literature


