Reengineering an Ada95-programmed Command and Control Information System by Using UML

Heinz Faßbender

Research Establishment for Applied Sciences
Research Institute for Communication, Information Processing, and Ergonomics
Neuenahrer Straße 20
D-53343 Wachtberg-Werthhoven
49-228-9435-640

fassbender@fgan.de

1. ABSTRACT

We describe the concepts and experiences we have made in an ongoing project by modeling and reengineering an experimental command and control information system which is nearly completely implemented in Ada95. For this purpose, we use the UML tool *Software through Pictures* from Aonix which includes a reengineering component that produces class diagrams from the source code. These diagrams serve as a starting point of a model of the complete system which itself serves as a base for modeling the most important requirements of the system. Furthermore, by refining the model through state-transition diagrams and redesigning the system the UML tool allows an automatic code generation for components which will be implemented in an other language like Java. Hence, the result of the reengineering process should be a more structured system, where proprietary solutions are replaced by commercial products (e.g. an application server) as much as possible.

1.1 Keywords

Reengineering, modeling, UML, case tools

2. INTRODUCTION

In the last decade a huge amount of effective and efficient methods for developing distributed information systems have been developed and standardized. The *Common Object Request Broker Architecture (CORBA)* [5] and *Enterprise Java Beans (EJB)* [7] are two examples for those standards. In the specifications of those standards it is often promised that their application would lead to better structured systems which are more safe, better extendible by new applications, and better maintainable.

Beside those methods a lot of visual modeling annotations are developed to form an abstraction from the source code and to illustrate the systems. Thus, it helps to manage the complexive structures of information systems. All those visual modeling annotations have been unified in the *Unified Modeling Language (UML)* [5]. This unification leads to a nearly formal annotation which is familiar to most of the developers. Furthermore, there exist a lot of case tools which support this annotation. That is why the application of UML also supports the development process of information systems, because the case tools often include an automatic code generation component.

These are two reasons, why we have decided to check the applicability of the new methods and standards in the most important research topic of our department, namely the context of command and control information systems.

Since we have a lot of experience in developing command and control information systems and we also have developed an experimental integration platform for command and control information systems **INFIS**, we also want to compare the new methods with those ones used to program INFIS in its old version.

For this purpose, we describe a reengineering process for INFIS. In particular, INFIS serves as the German testbed for interoperability tests in the *ATCCIS study* [2] and the *Multilateral Interoperability Programme (MIP)* [4]. The reengineering process is based on modeling the existing system by UML. In modeling the case tool *Software through Pictures* from Aonix is used for producing UML

diagrams from the existing code, as well as producing code from the redesigned diagrams.

In particular, this paper will not include a description of modeling the different phases in the development of a new command and control information system. Instead, it describes how to model and reengineer an existing command and control information system.

The paper is structured as follows. In Section 3 the global structure of INFIS is described by a self-defined annotation and the usage of UML is motivated. We present our reengineering process for INFIS in Section 4. This process determines the structure of the following sections. Modeling INFIS' high-level structure, its low-level structure, and their combination will be described in Sections 5, 6, and 7, respectively. In Section 8 the dynamic model of INFIS is illustrated. Finally, we complete the modeling process by extracting the requirements from the models and by modeling them by use case diagrams in Section 9. Furthermore, Section 9 contains a discussion about how the developed models can serve as starting point for adding new applications to the system. The paper is finished with some conclusions and some topics of future work in Section 10.

3. INFIS

INFIS is structured as a three level architecture (cf. Figure 1) of at least one **Domain**. Since the system is a multi-user system, it consists of a finite amount of graphical user interfaces **GUIs**. They are implemented as a Java Applet. The number of GUIs that are active, maybe arbitrary. Since the GUIs are implemented as a Java Applet, the system can be used totally platform independent with the following two minimal requirements to the resources of the clients:

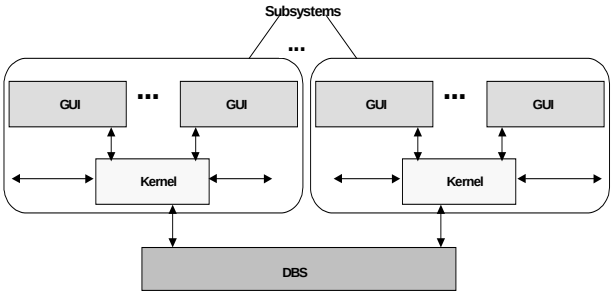

Figure 1: A Domain of INFIS

1.) Only an internet browser and

2.) a connection to the internet or an intranet are required.

The information of a Domain is stored in a database system **DBS** corresponding to the ATCCIS data model which is the main requirement for realizing the interoperability to other ATCCIS-conformant command and control information systems. The third tier is called the **Kernel**. Together with its connected GUIs it is called a **Subsystem**. The Kernel implements the application logic of the Subsystem and the connection to other Subsystems, as well as to other Domains. In particular, it implements the ATCCIS replication mechanism. Furthermore, the interfaces between the GUIs and the Kernels are implemented by the CORBA standard [5] which also supports the platform independent access to the system. The interface between a Kernel and the DBS is realized by a proprietary implementation that supports the concurrent access to the DBS.

The most interesting part in modeling the system is the **Kernel**. Its structure is illustrated in Figure 2.

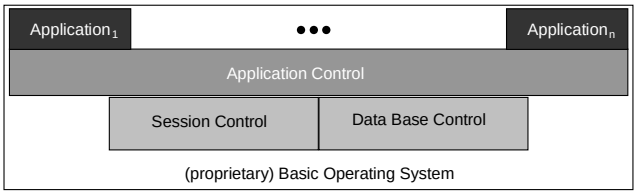

Figure 2: Structure of a Kernel

Every component of a Kernel is implemented as an Ada95 task [3]. The tasks communicate by messages. A task may communicate to every task which is in the illustration in Figure 2 close to it. I.e. every task except of the Applications can communicate with each other. **Applications** are positioned in the upper tier. They can only communicate with the **Application Control**, i.e. it is the border between the Applications and the rest of the Kernel. As an example of an Application we will use the initiation of the actual situation's display. A Session of a user is controlled by the **Session Control** and the access to the data base system is controlled by the **Data Base Control**. The **Basic Operating System** offers functionality of a multi-user and multi-process operating system. It is realized by a proprietary implementation to support the platform independent implementation of the system.

INFIS' architecture and the structure of a Kernel are illustrated in a self-defined annotation. It has only an illustrative and informal meaning, i.e. the models may be misunderstood by their readers and a lot of additional explanations are needed to give them a nearly formal meaning. Furthermore, there does not exist any software development tool which would support this annotation and there is no possibility to automatically produce code from these models with a case tool.

These are two reasons, why we have decided to develop formal models of INFIS by an annotation that fulfils the following requirements:

1. The annotation has to be understood by interesting people, because it has a nearly formal meaning.
2. The annotation must be standardized.
3. There have to exist software development tools that produce code from the models automatically.

The only existing annotation which fulfils all of those three requirements is the **Unified Modeling Language (UML)** that has been defined by Grady Booch, Ivar Jacobson, and James Rumbaugh from Rational Software Corporation [6], and that has been standardized by the Object Management Group OMG [5] which is a non profit consortium of nearly 400 companies. That is why we have decided to model INFIS by UML as the main task in our reengineering process.

In the rest of the paper we will define this process and we will illustrate the different phases of the process by presenting some models as examples.

4. INFIS' REENGINEERING PROCESS

The reengineering process for INFIS is illustrated as an UML-model itself in Figure 3.

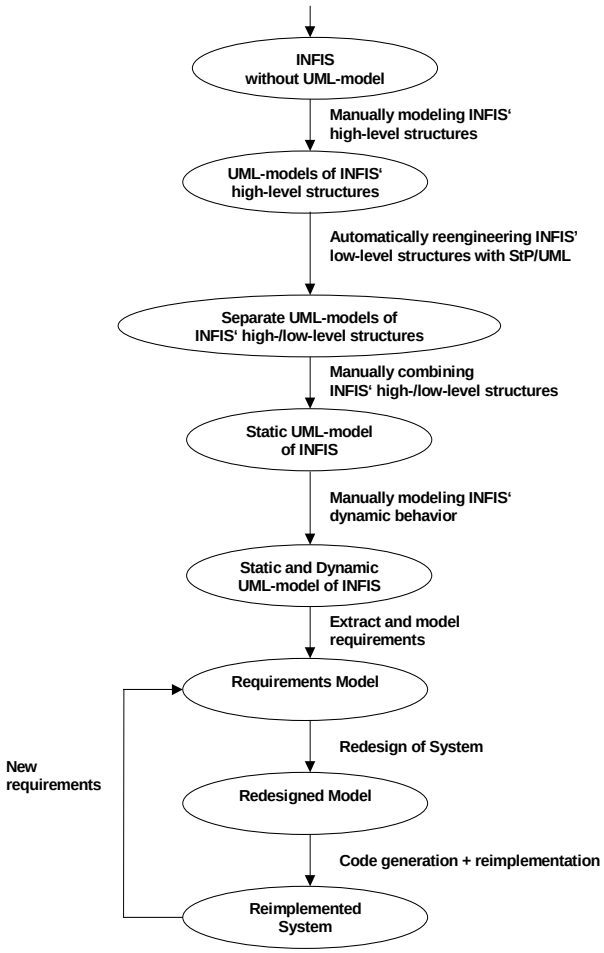

Figure 3: Reengineering Process for INFIS

The process starts with the existing version of INFIS which is only modeled by the self-defined annotation that is illustrated by Figures 1 and 2. First of all, the high-level structures of INFIS are modeled manually. After that, or in parallel to the modeling of the high-level structures, we use the reengineering component of the software engineering tool *Software through Pictures* from Aonix [1] for modeling INFIS' low-level structures that include the defined classes, their relationships, and the hierarchy of inheritance. After we have finished the modeling of INFIS' high-level structures, as well as its low-level structures, we combine these structures manually. After that, we develop dynamic models for applications which describe the interactions between the objects of the static model by computing the applications and the behavior of single objects.

Then we extract the requirements of the system from these models and develop the requirements model which finishes the process of modeling INFIS' old structure. The models serve as starting point for redesigning and reimplementing the system. Since the process is a never ending one, also the integration of additional applications which fulfill new requirements is considered.

5. UML-MODELS OF INFIS' HIGH-LEVEL STRUCTURES

We start the reengineering process by modeling INFIS' high-level structures. In opposite to the low-level structures which will automatically be illustrated by the reverse engineering component of the software development tool, the high-level structures have to be modeled manually.

In this section we present two UML-models of the high-level structures which correspond to the two models in Figures 1 and 2. The global architecture of INFIS is illustrated by the UML-model in Figure 4.

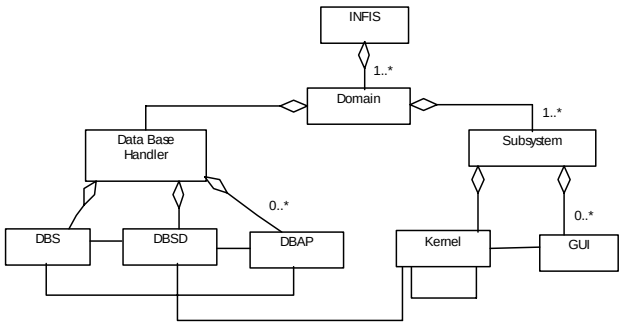

Figure 4: UML-Model of INFIS' Global Architecture

This model gives a more detailed description of INFIS' structure than the model in Figure 1. It contains the components of the architecture which are illustrated as *classes* in rectangles. The lines terminated by a rhomb, are called *aggregations*, i.e. the lines express *part of*-relations. For example INFIS consists of at least one (Notation: 1..*) Domain which itself consists of at least one Subsystem and

exactly one (This is the default value, if the end of an association or aggregation is not qualified.) Data-Base Handler, etc. The pure lines are called *associations.* An association between two classes indicates that there may be a message exchange between those classes.

Notice that the Data Base Handler, the DBSD, and the DBAP are components which implement the concurrent access to the Data Base System DBS.

The structure of a Kernel is illustrated as UML-model in Figure 5.

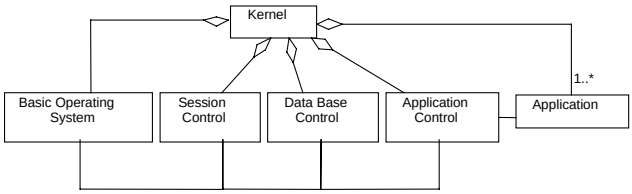

Figure 5: UML-Model for a Kernel

As described in Section 3 and illustrated in Figure 2, a Kernel consists of (*aggregations*) a Basic Operating System, a Session Control, a Data Base Control, and an Application Control which are associated to each other (*associations*). Furthermore, at least one Application belongs to the Kernel. The Applications are associated only to the Application Control.

If the UML-model in Figure 5 is compared to the model of the Kernel in Figure 2, then the difference of the expressive power between the two modeling annotations becomes obvious.

Beside manually modeling INFIS' high-level structures, its low-level structures can be modeled automatically. The result of this phase of the UML-modeling process for INFIS will be illustrated in the following section.

6. UML-MODELS OF INFIS' LOW-LEVEL STRUCTURES

For modeling INFIS' low-level structures, i.e. the classes in the code, their associations, and the inheritance hierarchies of the system we use the reengineering component of the software development tool *Software through Pictures* from Aonix. We have chosen this tool, because we also use the Ada95 development tool *Object Ada* from the same company. Software through Pictures produces a lot of class diagrams and class tables that include additional information about the classes of the system. Because of the syntactical structure of Ada95 programs which is not pure object oriented (compared to Java programs), the classes in the produced models only consist of attributes. Thus, the methods have to be manually added to the classes of the models. Nevertheless, the automatic generation of the models saves a lot of time for modeling and it reduces the number of mistakes in the models.

To give an impression of the result of the reengineering component of Software through Pictures, a part of the inheritance tree is shown in Figure 6. The classes of INFIS are represented by the nodes of the tree. The topmost node represents the class called **node**. It is the fundamental class of INFIS. There exist up to 1000 derivations from this node.

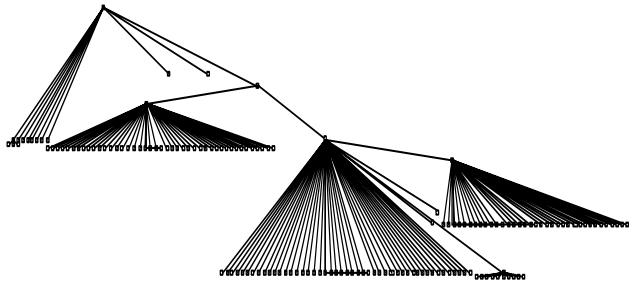

Figure 6: Inheritance Tree for the Application Control Components

In Figure 7 we present a more detailed result of the reengineering tool. Notice that the diagram in Figure 7 is not a part of the diagram in Figure 6 which includes the inheritance dependencies whereas Figure 7 only includes associations between classes and packages.

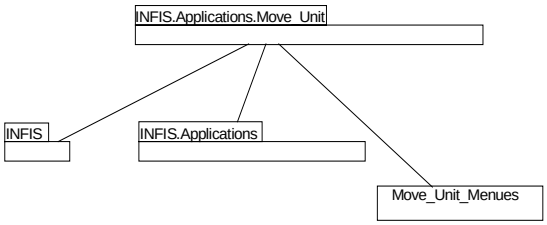

Figure 7: Detailed Result of an Automatic Generated UML-Model of the Low-Level Structure

This UML-model illustrates the associations of the INFIS application **Move_Unit** that allows to give a unit a new position on the situation display. The application **Move_Unit** is modeled as the package *INFIS.Applications.Move_Unit*. This results from the mapping of Ada95 programs to UML. It is associated to the parent packages *INFIS* and *INFIS.Applications*. Furthermore, it is associated to the class *Move_Unit_Menues* which stores the name and the coordinates of the unit that should be moved.

After the presentation of the UML-models of INFIS' high-level structures in the previous section and the illustration of the results of the application of the reengineering component we will explain how to integrate the UML-models of the high- and low-level structures.

7. COMBINATION OF UML-MODELS OF HIGH- AND LOW-LEVEL STRUCTURES

During reengineering of the low-level structures we found out that the classes in the high-level structures are

implemented by a huge amount of classes and packages in the code. That is why we have to connect the abstract classes in the high-level models to the implementations of their representations in the low-level models. Since the specification of UML 1.3 only allows that a package implements an interface (and not a class), the connections are realized via interfaces. For example, if we consider the high-level model in Figure 5 and the low-level model in Figure 7, we identify the package *INFIS.Applications* the name of which immediately indicates that it represents the implementation of the abstract class *Application* in Figure 5.

Then we simply combine the two models by associating the abstract class *Application* in Figure 5 with the package *INFIS.Applications* in Figure 7 via the Interface *Application Interface*. The result of this combination is represented in Figure 8 where also another low-level model that represents the implementation of the application *INFIS.Applications.Lokalis* is added.

through Pictures which allows to switch into the substructures of packages, as well as of classes.

8. DYNAMIC MODEL OF INFIS' APPLICATIONS

Up to now only the process of modeling the static structure of INFIS has been described. But, UML also offers some annotations as **sequence diagrams, collaboration diagrams, state-transition diagrams,** and **activity diagrams** to model the dynamic behavior of the system. We will explain the process of modeling INFIS' dynamic behavior by modeling the *computation of an application* through the different components of INFIS' static structure.

For this purpose, we use the modeling technique of *sequence diagrams* that is also used for modeling business processes. Sequence diagrams are the best choice, because

collaboration diagrams subsume the interactions between objects by abstracting from the order of the interactions. Since the order is an important fact in describing the

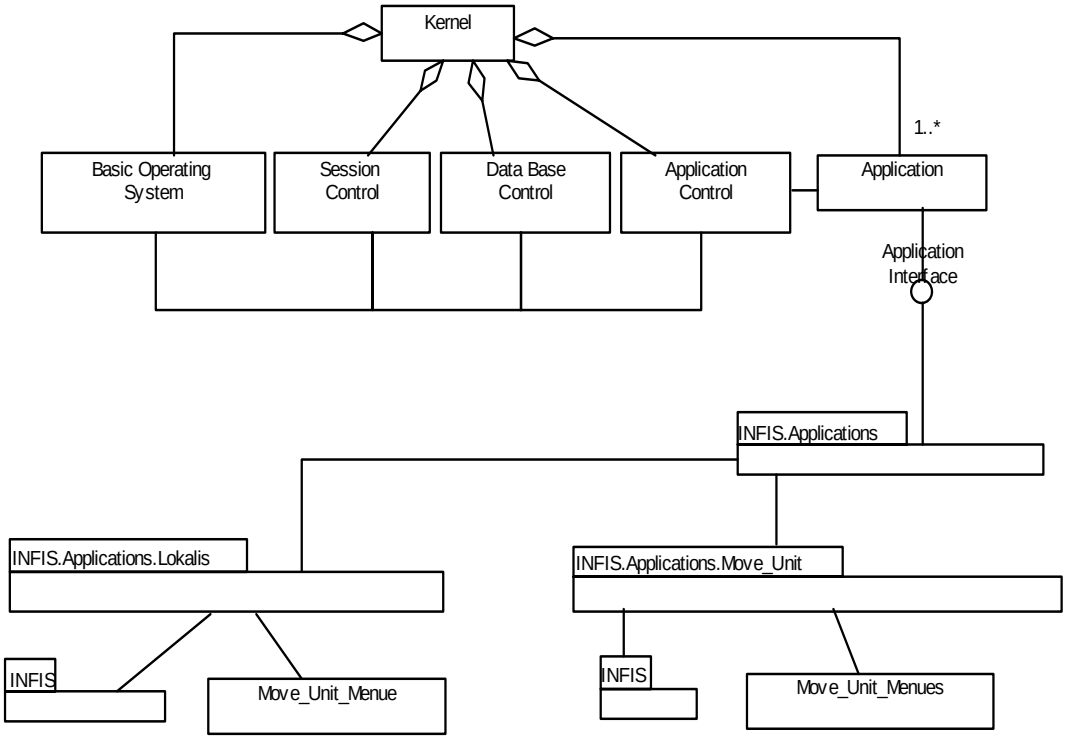

Figure 8: Combination of UML-Models of INFIS' High- and Low-Level Structures

The immediate integration of all UML-models of INFIS' low-level structures into packages of INFIS' high-level structures would obviously lead to unreadable illustrations. To solve this problem, we use the functionality of Software

dynamic behavior, collaboration diagrams are not as appropriate as sequence diagrams.

state-transition diagrams illustrate the reactions of one single object on events. They are not appropriate for modeling the interactions between objects. Nevertheless,

they will be a part in the model of INFIS' dynamic behavior.

activity diagrams are very close to sequence diagrams. But they are designed for modeling parallel processes.

In Figure 9 we present the UML-model for the computation of a general application.

notified about this permission. Then the user may write data to the template which is transferred to the application. The rest of the Application's behavior depends on the Application.

As mentioned before, beside the sequence diagrams the specific behavior of the objects are illustrated by **state-**

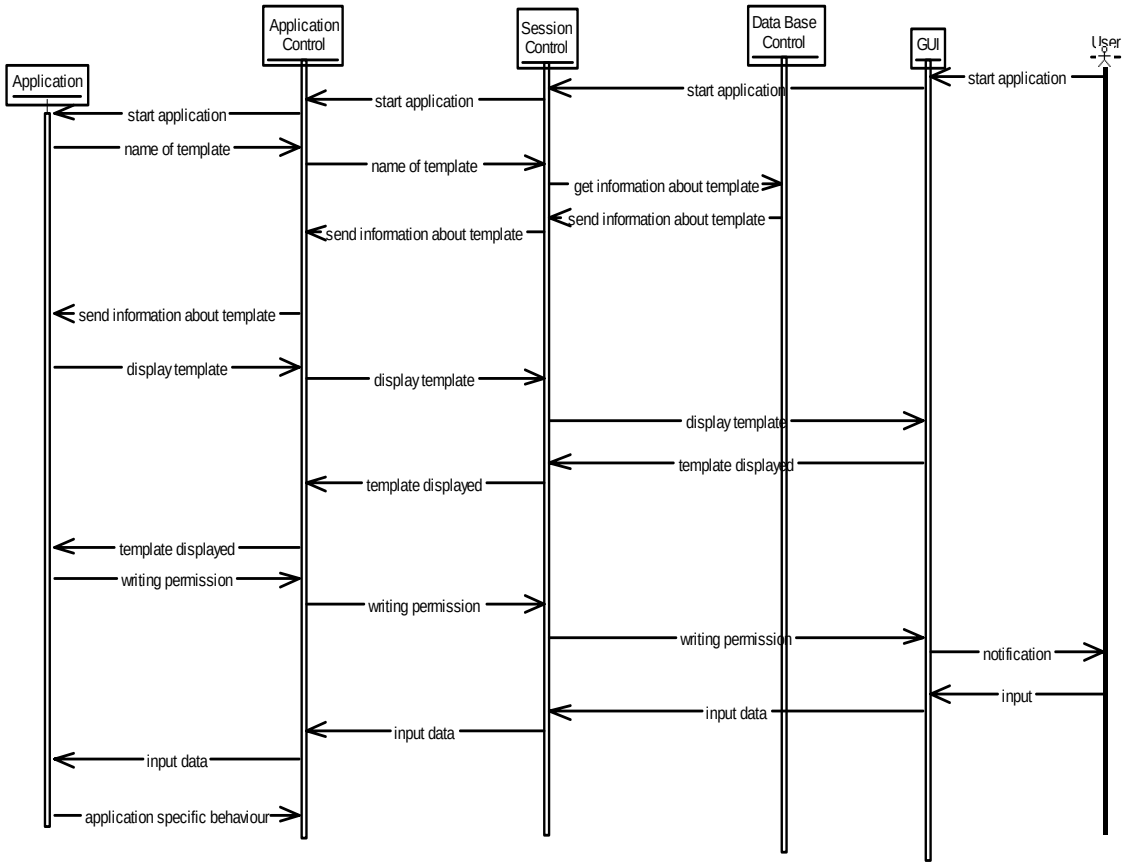

Figure 9: UML-Sequence Diagram for the Computation of an Application

In the sequence diagram there is one bar for every object and one bar for the user, if the user is involved in the process. The time axis is vertically from top to bottom.

When the user starts an Application, the GUI, the Data Base Control, the Session Control, and the Application Control are still running, but the Application object does not exist. Then the Application's initiation is sent from the GUI via the Session Control to the Application Control which starts the Application by creating an Application object. The information of the associated template is recalled from the Data Base by the Data Base Control and the template is displayed by the GUI which sends a confirmation to the Application. After that, the user's writing permission is sent to the GUI and the user is

transition diagrams like the one in Figure 3. We omit the presentation of such a diagram, since it would be too detailed in the context of this paper. Nevertheless, by the detailed description in state-transition diagrams we are able to use a specific feature of Software through Pictures that automatically transforms the information in those diagrams beside the information of the class diagrams into code. Thus, beside its illustrative character, the models serve as a starting point for designing and extending INFIS by new Applications.

9. REQUIREMENTS MODEL AND ADDITIONAL APPLICATIONS

For completing INFIS' models a huge amount of requirements is extracted from the developed models.

These requirements are modeled by use case diagrams. Such a diagram which models the two applications which are illustrated in Figure 8, is illustrated in Figure 10.

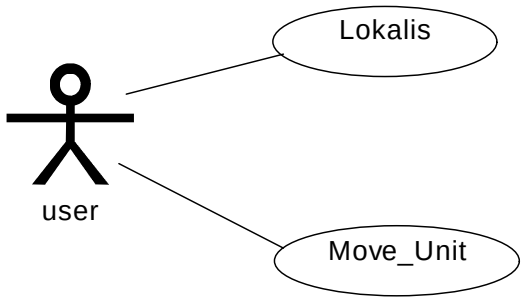

Figure 10: Part of the use case diagram

By this figure the strong correspondence between use cases and applications is expressed. In fact, this correspondence is responsible for the loop in the reengineering process in Figure 3.

After redesigning and reimplementing the system, new requirements can be considered. They will be modeled by new use cases in the requirements diagram. Furthermore, a new component for implementing the requirement as an application will be designed. For this purpose, the static diagram in Figure 8 is extended and the dynamic behavior of this component is modeled by a sequence diagram and state-transition diagrams for each of the objects of the component. Then code frameworks are automatically produced by Software through Pictures and the frameworks are completed by manually implementing the rest of the application.

10. CONCLUSION AND FUTURE WORK

We have presented a process for reengineering an existing experimental command and control information system which is based on modeling the system by UML. In the process we have used the software development tool Software trough Pictures from Aonix for producing UML diagrams from the existing code, as well as producing code from the redesigned diagrams. The defined process consists of a combination of automatic model generation by the reengineering component of Software through Pictures and manually constructed models. Furthermore, it consist of an integration of the two types of models and of modeling the dynamic behavior of Applications by sequence diagrams and state-transition diagrams.

Additionally, system requirements are extracted from the static and dynamic models and they are modeled as use case diagrams themselves. These part of reengineering can be called the *backward engineering part*, because it starts from a system without UML model and results in a complete model of the system, as well as of the requirements of the system.

The other part of the process can be called the *forward engineering part*. In that part with start with redesigning the system. Thereby, we observe that most of the system's components implement tasks that only manage the multi-user and multi-process behavior of the system. But, nowadays this functionality can be bought by commercial products as they are standardized, e.g. as an application server in the architecture of Enterprise Java Beans (EJB) [7]. Hence, we decided to redesign the system in such a way that the tasks that are positioned in the lower levels of INFIS' Kernel in Figure 2, are replaced by an application server and the applications are implemented as EJBs.

The redesign also leads to an easier implementation of applications, because the EJBs only have to communicate with the application server and with other EJBs.

This reimplementation of the applications as EJBs will be started in the following weeks and we will compare the new techniques with the one in the old implementation.

11. REFERENCES

[1] http://www.aonix.com

[2] *Army Tactical Command and Control Information System* (permanent), SHAPE Policy & Requirements Division, Mons (Belgium)

[3] Barnes, J.: *Programming in Ada95*. Addison-Wesley, 1995.

[4] http://www.dnd.ca/dlcspm/mip/index.html

[5] http://www.omg.org

[6] http://www.rational.com

[7] http://java.sun.com/products/ejb/