Combining OMT with a Prototyping Approach[1]

Wilhelm Hasselbring

Department of Computer Science, University of Dortmund, D-44221 Dortmund, Germany

Andreas Kröber

Informatik Centrum Dortmund, D-44227 Dortmund, Germany

Using the OMT software engineering methodology, a designer starts by identifying objects from the informal requirements specification. By identifying the relations between these objects, a global object model can be produced. Then, in an iterative process, this initial model is refined into an actual program. In this paper, we argue that the OMT methodology does not really help to analyze the requirements of a *new* proposed parallel system, and suggest combining OMT with the ProSet-Linda prototyping approach to requirements analysis of parallel systems to overcome some of the problems. The experience with this combination for the analysis of the requirements on a specific parallel system — a configurable hospital communication server — is discussed.

1 INTRODUCTION

Prototyping is used in the early phases of software development for requirements analysis, risk reduction, specification validation, and increased user acceptance of software systems (Budde et al., 1992). Some experience with developing *sequential* software systems using prototyping has been made (Gordon and Bieman, 1995). In particular when combining prototyping with an evolutionary software development process, not only the quality of the product, but also the development process itself can be improved (Lichter et al., 1994). However, software prototyping is not only concerned with the rapid development of user interfaces, but also with the functionality of planned systems (i.e. with developing algorithms). To be useful, prototypes must be *built* rapidly, and designed in such a way that they can be *modified* rapidly. Therefore, prototypes should be built in very high-level languages to make them rapidly available. ProSet-Linda is such a very high-level language which has been designed for prototyping *parallel* systems (Hasselbring, 1998).

The Object Modeling Technique (OMT) by Rumbaugh et al. is a popular object-oriented analysis and design methodology (Rumbaugh et al., 1991). The OMT notation combines an extended entity-relationship diagram (the *object model*) with state-transition diagrams for each object (the *dynamic model*) and some data-flow diagrams of processes that can be distinguished within the application's functionality (the *functional model*). On account of the natural concurrency of objects in this object-oriented technique, at first glance it seems to be a good technique for analyzing the requirements on parallel and distributed systems as they are required in hospital communication, which is the application domain of the presented project.

However, OMT still has some weaknesses:

- The OMT notation does not provide much support for requirements elicitation. It is rather a design notation.

[1]Preprint of a paper to appear in *The Journal of Systems and Software*.

- OMT's strong points are the data modeling features using an extended entity-relationship model, which is called *object model*. Unfortunately, OMT does not support hierarchical structuring of object models. Therefore, we extended the object model with hierarchical class definitions to support the specification of complex models.
- OMT performs badly when specifying real time and communication problems. The resulting problems are discussed below.

Prototyping is good for requirements elicitation and object-oriented modeling is good when the problem is well understood. To combine the advantages of both approaches, we distinguish between *first* and *extended* requirements analysis in the presented project. The first analysis provides an initial OMT model. For the hospital communication server, only basic structures could be identified, because a sensible functionality was not known at this stage of the development process. To evaluate and extend this initial model, ProSet-Linda has been used for prototyping. This extended requirements analysis through prototyping serves to refine the OMT model incrementally in several iterations. Below, one such iteration is presented. Essentially, OMT has been used to specify the insights gained through prototype evaluation in this project.

2 REQUIREMENTS ANALYSIS FOR A HOSPITAL COMMUNICATION SERVER

This section reports on the experience made with the combination of OMT with the ProSet-Linda prototyping approach for analyzing the requirements on a configurable hospital communication server (Hasselbring and Kröber, 1996).

2.1 The Background

The Group for Software Technology at the University of Dortmund is currently developing several components for hospital information systems in close cooperation with local hospitals. This includes the management of electronic patient records, clinical documentation, therapy control, and computer support for coordinating work within a hospital among other things. One central problem is the inevitable need to integrate legacy systems with modern information systems. As a first step towards this direction, the requirements on a configurable hospital communication server, which is based on the HL7 protocol, have been analyzed.

2.2 The HL7 Protocol

Health Level Seven (HL7) is an evolving standard for the electronic data exchange in hospitals. It is based on layer seven of the ISO/OSI-protocol hierarchy (Hammond, 1993). HL7 covers various aspects of data exchange in hospital information systems, e.g. admission, discharge and transfer of patients, as well as the exchange of analysis and treatment data. HL7 is a de-facto standard for data exchange between commercial systems for hospitals (Hammond, 1993).

An HL7 message is a string, which contains mandatory and optional segments. These segments consist of several fields. The syntax of version 2.2 of HL7 messages is defined informally in (HL7 Group, 1994). To gain an insight into the classification of the HL7 message types, we first analyzed the informal description of HL7 from (HL7 Group, 1994) and constructed a classification hierarchy with OMT's notation for the object model. An extract of this classification hierarchy is displayed in Figure 1.

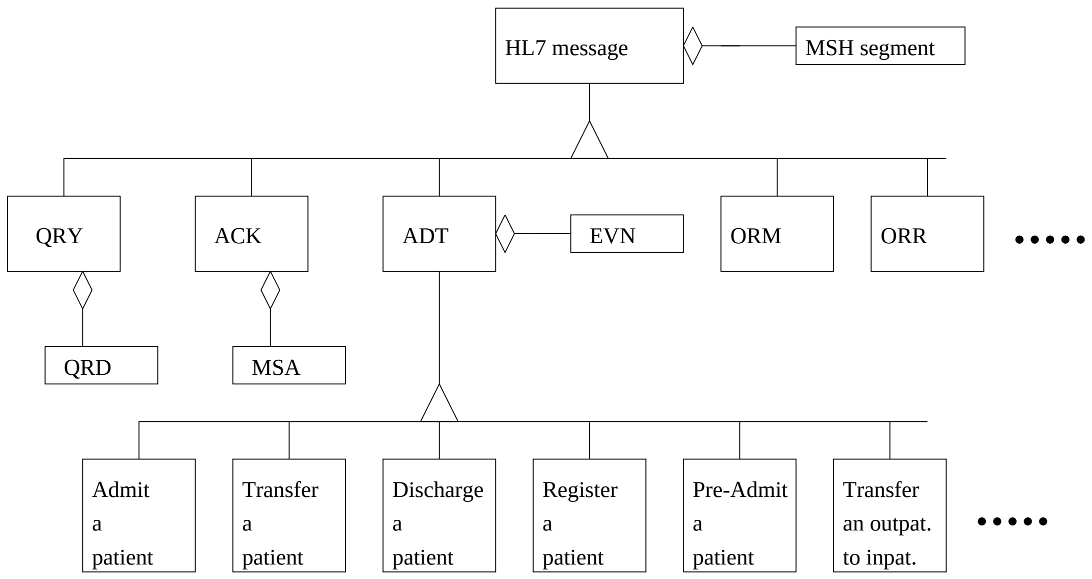

Figure 1. An extract of the HL7 message classification hierarchy. Rectangles are the OMT symbols for classes, triangles identify inheritance relations, and diamonds indicate part-of relations.

Unfortunately, the HL7 protocol does not cover all requirements of hospital applications. Many issues are not addressed, e.g. the exchange of images. Additionally, HL7 does not provide the option to send a message to a group of receivers (multicasting). As long as the HL7 group does not remove these limitations, a communication server might be used to compensate some of them.

The HL7 protocol has been designed to standardize the data transfer within hospitals. The goal of the presented project is to develop a configurable hospital communication server which allows the transfer of data between the distributed components of a hospital information system. This server needs the capability to integrate modern systems that communicate according to the HL7 protocol, as well as legacy systems that communicate through some non-standard protocol. It should be easy to tailor the server to fit in different hospital configurations. In addition, solutions to some of the still existing shortcomings in the HL7 protocol, e.g. the lack of support for *multicasting* of messages to a group of receivers, should be found. The essential goal of this project is to allow us to create integrated hospital information systems from heterogeneous components including legacy systems and modern information systems. Such a complex application requires a systematic development process.

For the initial requirements analysis and design, OMT is used. As we were not able to analyze the requirements exhaustively with OMT, prototypes have been implemented and evaluated with various scenarios for data exchange in a hospital information system to extend and modify the initial model.

2.3 Initial Requirements Analysis with OMT

One important concern in the specification of a hospital communication server is the configurability. The operation of the server in different configurations requires high flexibility of its interfaces. The server needs the capability to handle the different HL7 protocol versions (versions 2.1 and 2.2) and their regional adaptations, as well as vendor-specific protocols. Figure 2 displays a possible configuration of a hospital communication server. In this scenario, a laboratory, a radiology, a ward, an administration, and a pharmacy application are connected by the server. The server consists of a cen-

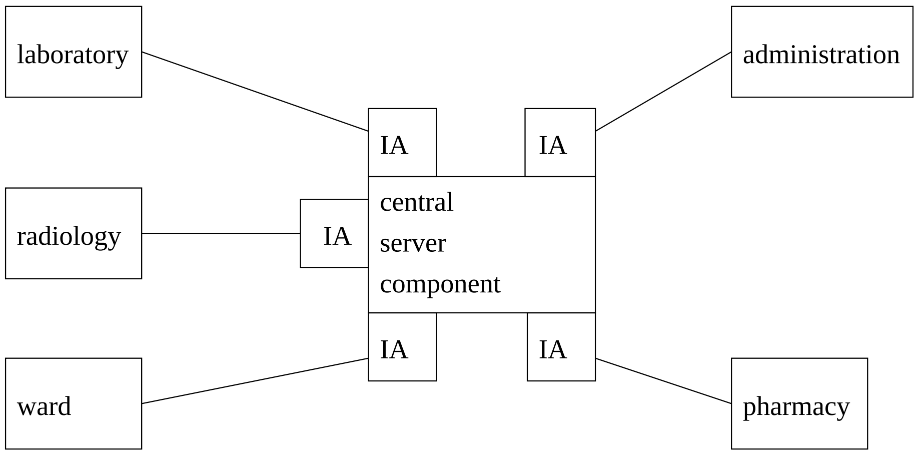

Figure 2. A possible configuration for the integration of a distributed hospital information system through a communication server. This is *not* OMT notation.

tral server component (CSC) and one interface agent (IA) for each application. The IAs transform messages from the individual applications into HL7 messages, if these applications do not communicate with the HL7 protocol. Internally, the CSC only handles HL7 messages to allow the easy replacement of applications. The IAs are components of the communication server. To replace an application, only the corresponding IA has to be adapted. Therefore, the IAs are considered to be part of the server's configuration.

With OMT, at first an object model is constructed to identify the classes in the problem domain to describe the static structure of the proposed system. Figure 3 displays an extract of the first object model, which shows the integration of applications with a hospital communication server. Every application communicates with one interface agent (IA). The IA appends an internal prefix to the message to let the server forward the message in an appropriate way. This internal prefix can be used, for instance, to specify priorities for specific messages. The server handles the messages according to his configuration. Several tables are used to configure the central server component (routing tables, protocol tables, multicasting tables, etc).

This initial modeling of the global data model was rather easy. During the refinement of the dynamic and functional models, we quickly came to the assessment that the complete analytical modeling of this complex system with the OMT methodology is almost impossible, because the knowledge about a useful functionality was not sufficient: the evaluation of executable prototypes has been initiated to experimentally explore the requirements on the dynamic behavior. In principle, the OMT *notation* is sufficient to model such a complex system, but we see the necessity to extend the OMT *methodology* with prototyping for requirements analysis.

2.4 Requirements Analysis through Prototype Evaluation

With OMT, we had problems to analyze *all* the requirements. For instance, the analytical modeling of the precise requirements on the concurrent processing of multicasted messages and acknowledgements is hardly possible. To fill this gap, we combine object-oriented analysis with the ProSet-Linda prototyping approach for requirements analysis.

The configuration from Figure 2, which shows the integration of different hospital systems by a communication server, has been implemented with ProSet-Linda. In Figure 4, the parallel processes and their tuple spaces, which are used for inter-process communication, are displayed. This figure

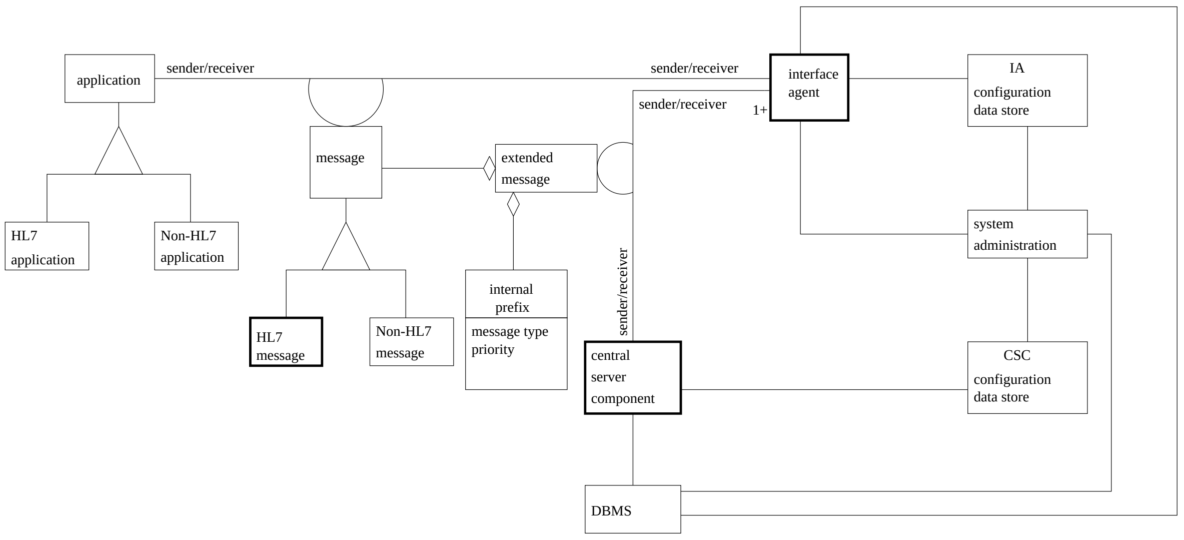

Figure 3. The first OMT model for the integration of applications with a hospital communication server. In OMT notation, link attributes are displayed as classes, which are connected by *loops* to its associations. The class 'message' is such a link attribute. Role names are written next to the association line near the class that plays the role. A complete description of OMT's notation can be found in (Rumbaugh et al., 1991).

illustrates the coarse architecture of the prototype. Each simulated application communicates with its interface agent through one tuple space for each direction. The IAs communicate through a common server tuple space with the central server component (CSC). In Figure 4, only one application is shown. Additional applications can be connected to the CSC through the common server tuple space.

The functional model in OMT consists of diagrams specifying the flow of data between processes within the application. In these models, a process is denoted by an ellipsis, while the data flow is represented by an arrow with a label indicating the sort of data that *flows*. Also, there are so-called data stores which merely serve to store data and intermediate results. They are denoted by two horizontal lines with the data store's name in between them.

The communication server is configurable through tables. A small example for the multicasting list in the set-oriented ProSet-Linda notation is:

```
multicast_list:={ ["Group1", {"App1","App2"} ],
                  ["Group2", {"App2","App3"} ] };
```

Each tuple in `rout_multicast_list` consists of a group identifier and a set of destination addresses. If an application sends a message to `"Group1"`, this message is multicasted to the applications `"App1"` and `"App2"`. The entries in this table are deposited by an administration process by means of ProSet-Linda's `deposit` operation at the tuple space for the CSC tables:

```
for entry in multicast_list do
  deposit entry at CSC_tables end deposit;
end for;
```

The CSC reads the information from this tuple space when the server needs it. This way the server may be re-configured online. The administration process is not displayed in Figure 4. The evaluation of the prototype is discussed below. Refer to (Hasselbring, 1998) for an introduction to the prototyping language ProSet-Linda.

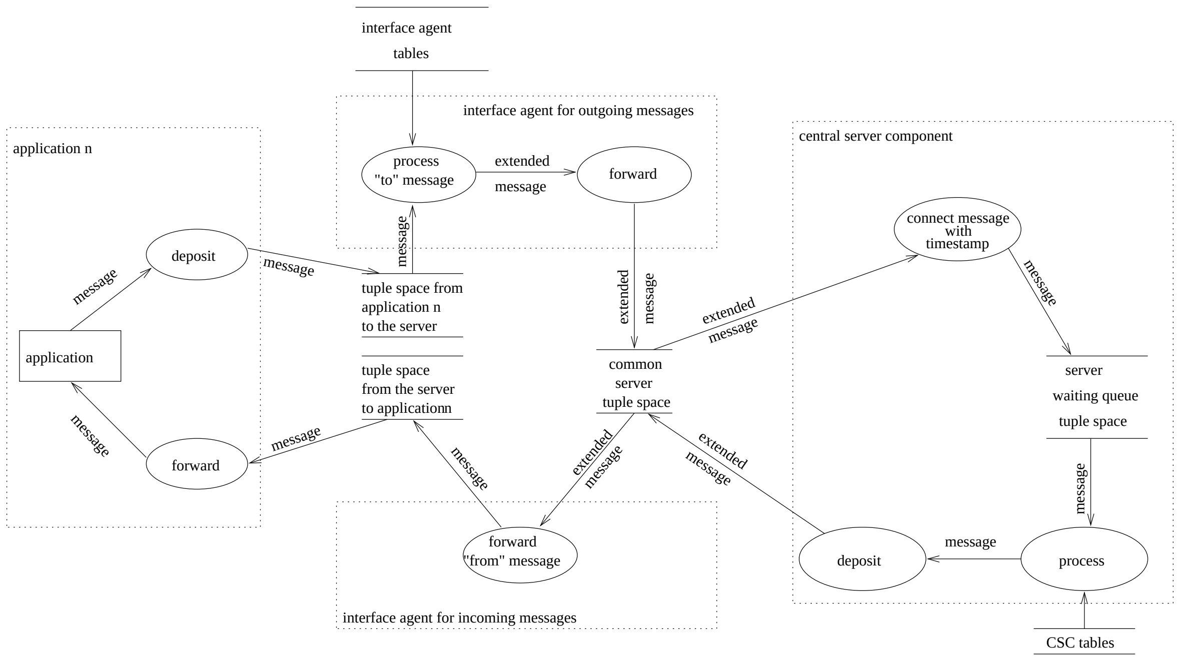

Figure 4. The inter-process communication within the prototype implementation. OMT's data-flow diagrams are used. The dashed frames are not original OMT notation. They are included to increase the readability. The IAs get their configuration, e.g. the protocol type, from the data store 'interface agent tables.' In the data store 'CSC tables,' the global configuration tables, such as routing list and multicasting list, are stored.

2.5 Extended Requirements Analysis and Design under Consideration of the Prototype Evaluation

The prototype implementation and evaluation offered detailed insights into the requirements to refine the initial model. Problems with the initial model, which have been discovered during the prototype's evaluation can now be avoided in the refined OMT model. During prototype evaluation, several doubtful design decisions were observed in the initial OMT model. For instance, the IAs only consisted of one process in the initial model (Kröber, 1996). In the prototype and the refined model, we split the IAs into two separate processes to avoid synchronization problems. With the initial model, each IA had to wait for all messages from both directions through active polling. The evaluation of the prototype showed, that this construction is error-prone and inefficient. Additionally, the analysis of the concurrent processing within the prototype implementation has shown that multicasting is not manageable with synchronous communication. Therefore, only multicasting with asynchronous communication is considered in the refined model (see (Kröber, 1996) for details).

This is the goal of experimental prototyping: Experimenting with ideas for different algorithmic variations and evaluating these ideas to make the right decisions as early as possible. In general, purely analytical evaluations are hardly feasible.

To construct large systems, it is advantageous to hierarchically structure systems. The OMT notation does not explicitly support modularization. Therefore, we extended the OMT notation for the object model by submodels. Each OMT class may be refined through such a submodel. Refined classes are displayed with a bold frame in Figure 3. The associations between external classes and the submodel must be displayed as *ports* in the submodel. The ports are connected to the submodel's internal classes. In principle, this form of hierarchy could be compared with macro mechanisms in programming languages. Each submodel could be expanded in its supermodel.

Figure 5 displays a refinement of the CSC from the global object model in Figure 3. The CSC consists of a waiting, a sending and two processing components. The simplified dynamic model for the message processing component is displayed in Figure 6 and the simplified dynamic model for the sending component is displayed in Figure 7. For each class in the object model, such a dynamic model is specified as a state-transition diagram. The global flow of events is displayed in Figure 8 in an event-flow diagram (Rumbaugh et al., 1991). The actions, which trigger events in other classes, are annotated at the links between these classes.

If the message processing component deposits an entry into the send list, the sending component sends the message. This communication is implicitly modeled in Figures 6 and 7. The event-flow diagram in Figure 8 makes this communication explicit. Obviously, the way communication is modeled with OMT's notation is rather informal. The needed synchronization cannot be modeled with OMT. Additional informal specifications are necessary. A detailed description of the prototype evaluation and the extended model is beyond the scope of the present paper and may be found in (Kröber, 1996).

3 RELATED WORK

Several other approaches which intend to overcome some of the shortcoming of the OMT methodology are discussed in the literature. In (Knowles and Collingwood, 1994), for instance, OMT has been combined with the PARSE methodology (Gorton et al., 1995), because the object model did not completely provide the necessary refinement required in order to express the final software design. OMT has been extended to include the additional considerations of parallelism by mapping the resulting object model to PARSE. The PARSE process graph notation allows to describe a parallel system in terms of a hierarchy of interacting components. These components are either passive function or data servers, or active control processes. Processes interact by message passing on designated communi-

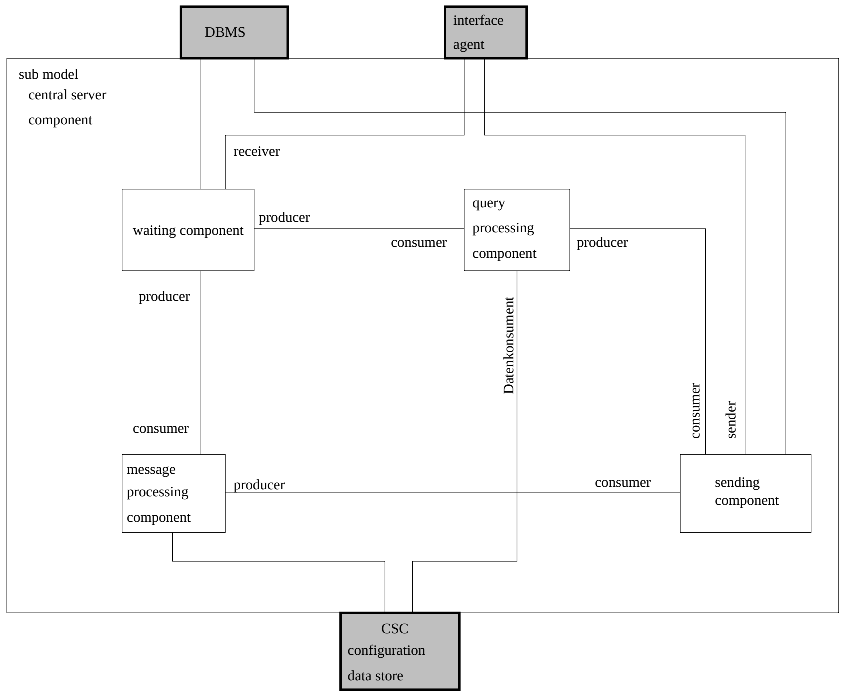

Figure 5. The submodel ’CSC.’ The ports are displayed with a grey background.

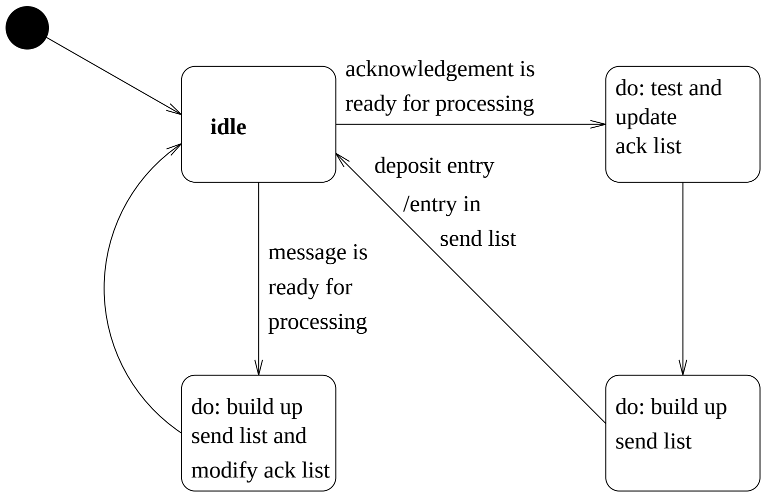
idle
acknowledgement is ready for processing
do: test and update ack list
deposit entry /entry in send list
message is ready for processing
do: build up send list and modify ack list
do: build up send list

Figure 6. The simplified dynamic model for the message processing component from Figure 5. With OMT’s state-transition diagrams, the start state is indicated with a black circle. An event name is written on a transition arrow. An action is indicated on a transition following the event name by a “/” character followed by the action’s name (Rumbaugh et al., 1991). The action may trigger an event with the same name. Here, the action ‘entry in send list’ triggers the corresponding event in Figure 7.

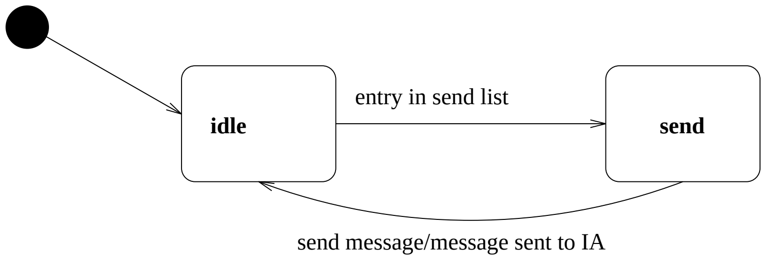

Figure 7. The simplified dynamic model for the sending component from Figure 5.

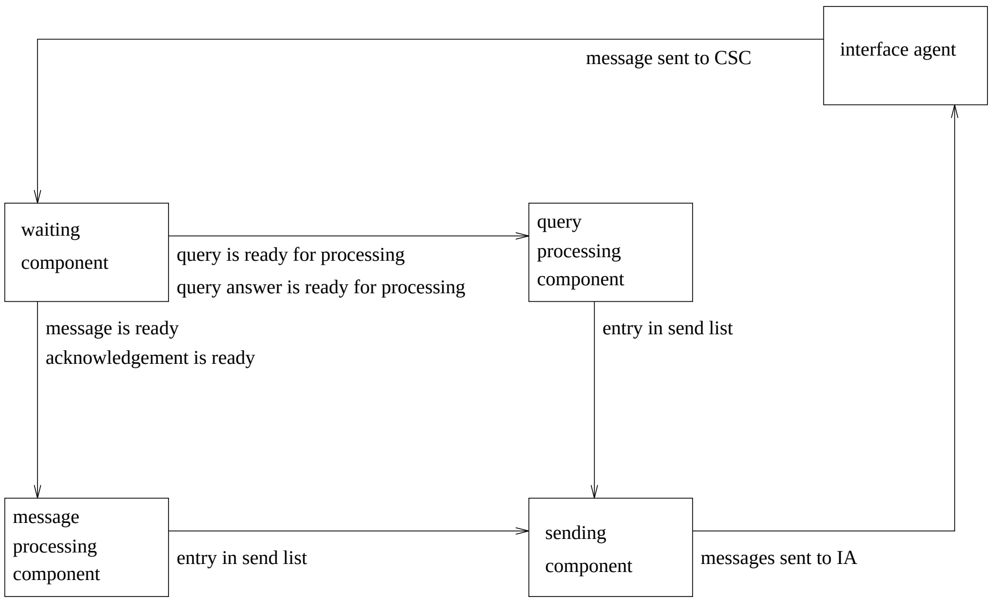

Figure 8. A small extract from the event-flow diagram for the hospital communication server.

cation paths. Essentially, OMT's dynamic model has been substituted by the PARSE process graph notation in this approach. Within the PARSE project, the integration of behavioral analysis techniques involving Petri-Nets for design validation has been explored through (manual) translation of process graphs into Petri-Nets to offer a path to formal verification (Gorton et al., 1995). With this approach, a form for prototyping could be based on Petri-Net simulation.

As we could see in the preceding section, OMT performs badly when specifying communication and synchronization problems, With OMT, the dynamic behavior is specified on an object-class basis by state-transition diagrams. The diagraming technique includes symbols for states (rounded boxes) and transitions (arrows between boxes). Usually, transitions from one state to another are triggered by internal or external events, which appear in labels at the arrows. An external event is an event triggered by an action of one object class, and affecting another object class. However, serious problems arise with the semantics of communication when using the OMT notation. There are no means other than textual circumscription in state-transition diagrams to describe, for instance, whether communication is synchronous or asynchronous. The event-flow diagrams do not really solve this problem, because the relations to the individual state-transition diagrams are not formally specified. One point to make is that OMT's suitability for developing *parallel* and *distributed* software is limited: while excelling in data modeling, its support to model communication is insufficient. As an aside, we mention that the Booch method (Booch, 1994) does incorporate some explicit communication notations in its object diagrams (e.g. synchronous vs. asynchronous communication). The Booch Method's object diagrams can be compared to some extend with OMT's event-flow diagrams, but on an object-instance level, not on the class level at it is the case with OMT. However, the semantics of these are still not formally defined. The forthcoming Unified Modeling Language (Rational Software Corporation, 1997) allows a more detailed specification of realtime and concurrency concerns, but still in a semi-formal way.

A formal approach to specify distributed systems with state-transition diagrams is the Requirements State Machine Language (RSML) (Leveson et al., 1994). To allow an adequate modeling of distributed systems, RSML extends state-transition diagrams by explicit interfaces and directed communication channels between individual state machines.

Other object-oriented methods have been used to analyze parallel systems (Knowles et al., 1995). However, to our knowledge, there exists no published work that explicitly combines an object-oriented analysis method with a prototyping approach for requirements analysis of *parallel* systems. Therefore, we combined the OMT methodology with the ProSet-Linda prototyping approach to gain some experience with this combination.

4 CONCLUSIONS

This paper presents some problems and solutions for the requirements analysis of a specific parallel system. A hospital communication server, which can be configured by means of various tables, was discussed to connect heterogeneous components of hospital information systems and to overcome some of the shortcomings of the HL7 protocol. Within the scope of the present paper, only some aspects of this application could be presented. We would like to comment on our experience with the employed analysis techniques.

A point to stress is the employed development process: The combination of OMT with a prototyping approach for requirements analysis of parallel and distributed systems. The observation to make is that OMT appears to be a good technique for the analysis of systems with *already known* properties, but for *new* systems, like a hospital communication server with many unknown requirements, experimental prototyping is a promising extension. The requirements analysis process can be depicted with this approach as illustrated in Figure 9.

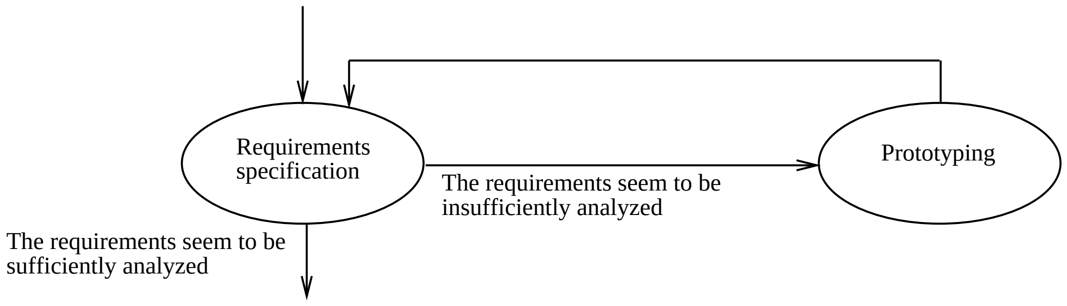
Requirements specification
Prototyping
The requirements seem to be insufficiently analyzed
The requirements seem to be sufficiently analyzed

Figure 9. The process for requirements analysis with the combination of OMT with a prototyping approach.

When the requirements seem to be sufficiently analyzed, the next phases in the development process can be started. The decision, whether the requirements are sufficiently analyzed and whether prototyping should be carried out depends on existing knowledge in the application domain. For a re-implementation of an old system with the goal to improve non-functional properties such as performance and reliability, prototyping will not be very helpful for the requirements analysis. For the implementation of a new system which is supposed to solve new tasks, it will be helpful to carry out several prototype evaluations. This was necessary for the presented requirements analysis on a hospital communication server.

The principal question arises whether the OMT methodology alone is at all usable for the requirements analysis of *new* systems. At least the notation only supports the design. The method to identify objects from the informal requirements specification is described informally in (Rumbaugh et al., 1991). There exists no explicit support for requirements analysis in OMT. With OMT, it is not easy to distinguish between analysis and design. Embley et al., for instance, argue that the existing object-oriented analysis and design techniques (including OMT) do not appropriately support analysis, because the distinction between analysis and design is not explicit with these techniques (Embley et al., 1995). In (Embley et al., 1995), a new analysis method, Object-Oriented Systems Analysis (OSA), is proposed which is exclusively designed for analysis. OSA specifications are executable (they have formally defined syntax and semantics). This way, prototyping is directly supported by this analysis method. In our requirements analysis, we still had to use two different methods for object-oriented analysis/design and for prototyping. Essentially, OMT has been used to specify the knowledge gained through prototype evaluation in our project. The high level of ProSet-Linda's constructs for parallel programming enabled us to rapidly develop prototypes of parallel programs and to experiment with parallel algorithms.

REFERENCES

Booch, G., *Object-Oriented Analysis and Design with Applications*, 2. edition. Benjamin/Cummings, Redwood City, California, 1994.

Budde, R., Kautz, K., Kuhlenkamp, K., and Züllighoven, H., *Prototyping — An Approach to Evolutionary System Development*. Springer-Verlag, Berlin, 1992.

Embley, D., Jackson, R., and Woodfield, S., OO Systems Analysis: Is It or Isn't It? *IEEE Software* 12, 19–33 (1995).

Gordon, V., and Bieman, L., Rapid prototyping: Lessons learned. *IEEE Software* 12, 85–95 (1995).

Gorton, I., Gray, J., and Jelly, I., Object based modelling of parallel programs. *IEEE Parallel and Distributed Technology Journal* 3, 52–63 (1995).

Hammond, W., Health Level 7: A protocol for the interchange of healthcare data, in *Progress in Standardization in Health Care Informatics* (G. D. Moor, C. McDonald, and J. van Goor, eds.) IOS Press, Amsterdam, 1993, pp. 144–148.

Hasselbring, W., The PROSET-Linda approach to prototyping parallel systems. *The Journal of Systems and Software*, to appear (1998).

Hasselbring, W., and Kröber, A., Requirements Analysis through the Combination of OMT with a Prototyping Approach (in German), in *Proc. GI-Fachtagung Softwaretechnik 96*, Softwaretechnik-Trends 16/3, 1996, Koblenz, Germany, pp. 105–112.

HL7 Group., Health level seven: an application protocol for electronic data exchange in healthcare environments, version 2.2. Technical report, Health Level Seven, Inc., Ann Arbor, USA, 1994.

Knowles, C., and Collingwood, P., Parallel software development using an object-oriented modeling technique. *Information and Software Technology* 36, 397–404 (1994).

Knowles, C., Collingwood, P., and Jelly, I., Evaluation of software engineering analysis techniques for parallel software, in *Proc. 21th Euromicro Conference*. IEEE Computer Society Press, Piscataway, N.J., 1995.

Kröber, A., Modeling a configurable HL7-based hospital communication server (in German), Master's thesis, Department of Computer Science, University of Dortmund, 1996.

Leveson, N., Hildreth, H., Heimdahl, M., and Reese, J., Requirements specification for process-control systems. *IEEE Transactions on Software Engineering* 20, 684–707 (1994).

Lichter, H., Schneider-Hufschmidt, M., and Züllighoven, H., Prototyping in industrial software projects — bridging the gap between theory and practice. *IEEE Transactions on Software Engineering* 20, 825–832 (1994).

Rational Software Corporation., The Unified Modeling Language. Documentation Set Version 1.0, Santa Clara, California, 1997.

Rumbaugh, J., Michael, B., William, P., Frederick, E., and William, L., *Object-Oriented Modelling and Design*. Prentice Hall, Englewood Cliffs, N.J., 1991.