A Case Study in Object-oriented Modeling and Design of Distributed Multimedia Applications[1]

Angelo Morzenti[1], Matteo Pradella[1], Matteo Rossi[1], Stefano Russo[2], Antonio Sergio[2]

[1] Dipartimento di Elettronica e Informazione, Politecnico di Milano,
Piazza Leonardo da Vinci 132, 20133 Milano, Italy, morzenti@elet.polimi.it

[2] Dipartimento di Informatica e Sistemistica, Università di Napoli Federico II,
Via Claudio 21, 80125 Napoli, Italy, Stefano.Russo@unina.it

Abstract. This paper investigates the use of object-oriented techniques for the specification and design of distributed multimedia applications (DMAs). DMAs are a class of software applications with a range of strong – often conflicting – requirements of dynamicity, interactivity, real-time synchronized processing of several media types, network distribution, high-performance, fault-tolerance, load balancing and security. The development of complex DMAs can benefit from the adoption of object design methods and distributed objects implementation technologies. The paper describes the use of two modeling approaches, based on the standard UML modeling language, and on the TRIO formal specification language, respectively. The problem of defining steps to move from the UML or TRIO specification to a CORBA IDL implementation is addressed too. An experimental distributed Video-on-Demand system is used throughout the paper as a case study.

1. Introduction

Distributed Multimedia Applications (DMAs) are a class of software systems that benefits from technological advances in several areas – from networking to image and other media processing – to provide end-users with highly interactive, real-time access to remote resources and/or to distributed collaborative processing. Examples of DMAs are computer-supported cooperative work systems, video conferencing systems, video-on-demand systems [1].

Key characteristics of DMAs are interactivity, multimediality (i.e., the processing of several media types), dynamicity, network distribution, high-performance and synchronisation between different media types (for real-time delivery of multimedia sources). The requirements of advanced DMAs often include also issues of fault-tolerance, load balancing and security. Further desirable features include user friendliness, network transparency, fault-tolerance and quality-of-service guarantee.

This paper investigates the use of object-oriented (OO) techniques for the development of a typical DMA, a Video-on-Demand system (VoD). Two specification approaches are discussed, based on the standard UML modeling language [2], and on the TRIO formal specification language [3], respectively. As distributed objects platforms, especially CORBA, are gaining increasing popularity [4], we also address the important issue of moving from the specification to the CORBA IDL definition of objects interfaces.

The paper structure is the following. Next two Sections summarize expected benefits of using OO techniques for DMAs, also in relationship to previous related work. Section 4 introduces the DiVA VoD case study. Sections 5 and 6 describe DiVA modeling with UML and TRIO, while Section 7 discusses strategies for moving from the specification to a CORBA implementation. Section 8 gives concluding remarks.

2. OO design of DMAs

OO software engineering offers a way to deal with software development which centres all phases, from domain modeling to software design and implementation, around the concept of object, as an entity that incorporates both a state and a behaviour.

Many OO analysis and design (OOA&D) methods appeared in the late ’80s and early ’90s, including Schlaer and Mellor, Coad and Yourdon, Booch, Rumbaugh, Jacobson. Recently, these last three authors have joined their forces in the effort to define a standard method. The result is a unified notation for analysis and design known as Unified Modeling Language [2].

UML provides a rich set of diagram types to describe static and dynamic aspects of the system under development, modeling user requirements, system components, their interactions and their internal behaviour. There exist commercial CASE support tools to produce UML diagrams and acting as software documentation repository. UML pays much attention to ease of use and to providing support to mature and engineering practices to software modeling. However it lacks formality, especially for modeling behavioural and real-time aspects.

[1] This work has been carried out partially with the financial support of the Ministero dell'Università e della Ricerca Scientifica e Tecnologica (MURST) under Project MOSAICO (Design Methodologies and Tools of High Performance Systems for Distributed Applications).

One formal language for OO specification of real-time software systems is TRIO [3]. The specification of a system in TRIO consists of a graphical and a textual part. The graphical notation is used to depict classes with their interfaces, and the connections between them. Classes may be simple or structured, i.e. composed of subclasses. The textual part describes, in addition to the above, time dependent and time independent items (constants, functions, predicates, propositions), and axioms, which are formulas in a temporal logic language. Using TRIO, the designer can specify formally the structural as well as the real-time aspects of the system to develop, combining sound OO abstraction mechanisms such as information hiding, inheritance and genericity, with the unambiguity of logic specifications.

As for implementation, in the last years a major paradigm shift in the area of client-server systems has been driven by the advent of distributed objects technology [4]. The Common Object Request Broker Architecture (CORBA) standard by the Object Management Group is one such middleware platform [5].

OO methods and distributed objects technology can facilitate the development of complex DMAs. On the one hand, OO design methods promote strong modularity, which is essential to DMAs [6]. They ease the identification of software components (objects) with clear boundaries, each responsible for providing in a transparent way the various system services and tasks, such as operating media hardware, managing network connections, implementing control policies, providing fault-tolerance and security, and interfacing end-users.

On the other hand, distributed objects technology claim to provide definite advantages in terms of development cost, management of heterogeneity and distribution of resources, location transparency, reuse of existing subsystems, software modularity, and ease of integration with Web technology.

Despite this, no dedicated methods exist today for the development of CORBA systems, as well as no technique is available for moving from an OO specification to a CORBA architecture. These are still open research issues. By means of a sample DMA application, this paper deals with the definition of methods to address such issues.

3. Related work

Previous work on the OO design of a DMA has shown the suitability of the PARSE methodology for this purpose [6]. PARSE is an object-based methodology supporting a systematic design refinement process, based on a graphical notation named *PARSE process graphs* [7].

The mapping from the PARSE design onto a message passing Application Programming Interface is also discussed in [6]. Here we are concerned with distributed objects target platforms.

4. The VoD case study

DiVA (Distributed Video Architecture) is an experimental distributed Video-on-Demand system under development at the Department of Computer and Systems Engineering of the Federico II University of Naples [8]. DiVA is an interactive client-server system for retrieval and real-time delivery of video sources from a remote distributed database; as such, it is a typical DMA.

The general structure of DiVA is depicted in Fig. 1. The video sources are distributed over the network in several sites, managed by *film server* objects. A central application server (*VideoServer*) is assumed to provide the user clients with the functionality of searching the distributed database for available films and selecting a video source. The client and film server computers then synchronize for audio/video real-time data transmission. Multiple user sessions may be active concurrently.

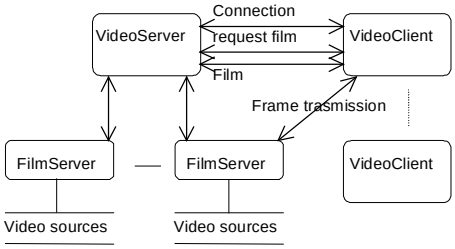

Fig. 1: The DiVA system.

5. DiVA modeling with UML

The first step in the OO development is to describe user requirements; these are captured in UML by means of *use cases* diagrams. A use case is a metaphore for describing a user interaction with the system[2]; use cases are obviously related to system functionalities. Fig. 2 shows the use case diagram for our VoD system. For brevity, further functionalities for other kinds of actors, such as configuration functions acted by the system manager, are not modeled here. The *extends* relationship in Fig. 2 means that ListRequest and FilmRequest are subcases of the more general ConnectionRequest use case.

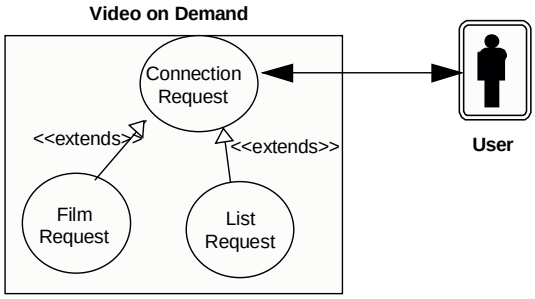

Fig. 2. DiVA UML use case diagram.

[2] Actually, *actors* are shown in a use case diagram, modeling roles played by users in typical interactions with the system.

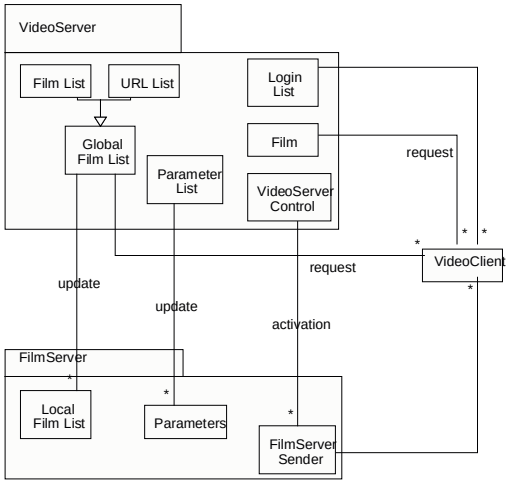

Fig. 3. DiVA UML class and component model.

The next step is to identify problem domain objects. These are captured in the UML *class diagram*. The set of classes of DiVA objects and their relationships (hierarchical and associations) are shown in Fig. 3. There are objects responsible for managing local and global directories of available clips, configuration and load parameters, and video frames transmission to the client.

As we move toward the design phase, the problem domain object model is refined in several ways. One concerns the inclusion of solution domain objects and responsibilities. In the development of DMAs, it is at this stage that concerns such as how to operate media and network hardware come into play. In DiVA, for instance, the FilmServer object is responsible for delivering audio-video data to VideoClient. For this purpose, DiVA uses a specialised software library, the Continuous Media Toolkit (CMT) developed at Berkeley University [9]. The use of CMT is encapsulated within proper objects created by FilmServer and VideoClient. A future version of DiVA will be based around the Java Media Framework library [10]. A good object-based design should provide the advantage of high locality of changes for such maintenance activity. Ideally, the replacement of CMT with JMF would be totally transparent to other methods and objects.

The refinement of the object model proceeds also in the direction of identification of system components. Objects with a strong degree of cohesion are grouped together, representing subsystems with clear boundaries. The UML *component diagram* captures this aspect. For distributed systems like DiVA, components should represent candidate subsystems with respect to partitioning and allocation. Clustering of objects in the DiVA design is shown for the sake of space on the object model (Fig. 3). Three subsystems are shown: FilmServer, VideoServer and VideoClient.

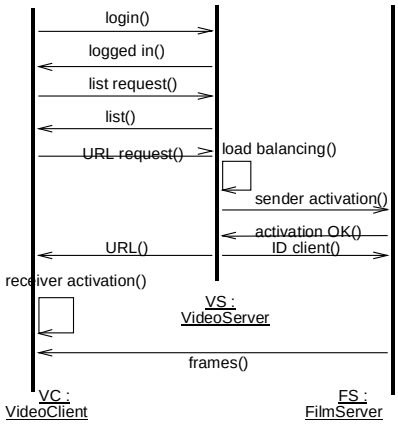

Fig. 4. UML sequence diagram.

Having identified system objects and components, the design proceeds to the definition of the objects' behaviour. The class diagram usually contains the classes' responsibilities, in terms of the main operations they offer. The external behaviour of objects is now further refined describing the interactions between them. UML provides *sequence* or *collaboration diagrams* to describe how groups of objects collaborate in some behaviour.

In DiVA, the flow of messages in a typical user session is shown in Fig. 4. Once a user request has been accepted, VideoServer interacts with FilmServer to start delivery of the film (sequence diagram is not shown here).

The OO design proceeds through the definition of the internal behaviour of each object in the system. For this purpose, UML provides *state diagrams*, a variation of David Harel's statecharts. Figure **5** shows the state diagram of a VideoClient object.

Drawing state diagrams is useful for specifying the dynamics, across use cases, of a single (non-trivial) object. They complement interaction diagrams, which are good at describing the behaviour of several objects in a single use case. Moreover, consistency checks can be performed (supported by CASE tools) with respect to the object's interface, since state changes happen only as a result of events that reach the object.

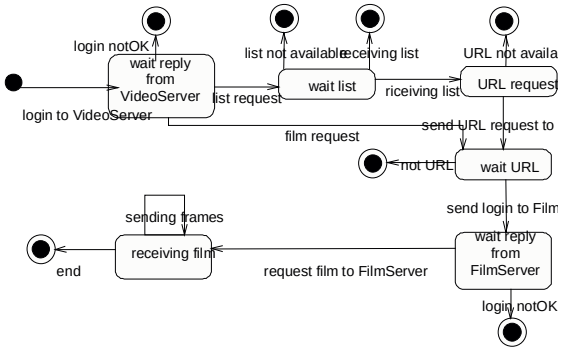

Fig. 5. State diagram of a VideoClient object.

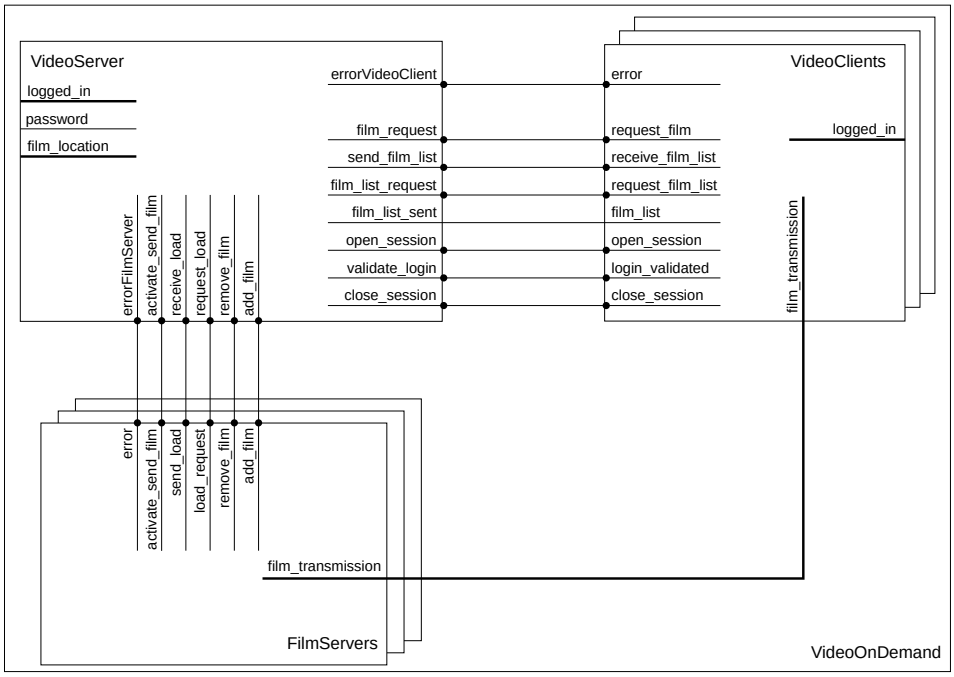

Fig. 6. DiVA TRIO class diagram.

6. DiVA modeling with TRIO

Figure 6 shows the class diagram of DiVA, using the TRIO notation. This notation is quite intuitive: the boxes represent the classes. A class logic item is depicted by a line into the class' box: this line continues outside when it's visible. Bold line stands for *state* items, i.e. items possibly holding in intervals of the time domain. Items represented with a dot are *event* items, i.e. items possibly holding only in insulated instants of the time domain. Arrays of classes are depicted by stack of boxes.

As we can see, the TRIO class diagram structure is almost identical to the one of Fig. 3. The main difference resides in the description of operations: one operation is usually represented by a complex set of logical items and relations between them. So the number of connections in the TRIO class diagram is naturally greater than the number of arrows shown in Fig. 6. As we will see, the TRIO/CORBA methodology permits the passage from "simple" logical items to complex CORBA concepts like operations and attributes.

The brief description of the TRIO specified system is the following. A *video client* sends its identification data (login, password) to the *video server* to open a session (event 'open_session'); if the client is authenticated ('login_validated'), it can then issue requests to the video server, to obtain the list of available films ('request_film_list'), or to receive a specific film ('request_film'). When the video server receives a request for an available film, it determines which is the *film server* that will transmit the film to the video client; it then requests ('activate_send_film') the film transmission from the film server to the video client. Like for UML, in the TRIO specification film transmission has not been modeled in its details: we represented that fact that multimedia data are exchanged between a film server and a video client by means of a state ('film_transmission'), which describes when data (frames) are flowing. For load balancing reasons, the video server can ask a film server which is the work-load it is carrying ('request_load'). A film server notifies the video server when a film is added/removed from its local database ('add_film', 'remove_film'). When a video client has no more requests to issue, it closes the session opened by the video server ('close_session'). Errors are signaled ('errorVideoClient', 'errorVideoServer') when illegal operations are performed.

Fig. 7 shows a sample axiom taken from the class VideoServer. The intuitive meaning of this formula is the following. If a video client vcID issues a valid film list request to the video server (i.e. if the request is associated with a valid session identifier sID), then either the film list is sent to the video client, or error 'ListNotAvailable' is signaled, within a time interval of RESP_MAX_DELAY. RESP_MAX_DELAY is a constant which depends on the system and the used ORB.

Answer_to_a_film_list_request_1

$$\text{film\_list\_request}(vcID, i, sID) \wedge \text{logged\_in}(sID) \Rightarrow$$
$$\text{WithinF}\,(\text{send\_film\_list}(vcID, i) \vee \text{errorVideoClient}(vcID, i, \text{ListNotAvailable}),\ \text{RESP\_MAX\_DELAY})$$

Fig. 7. A TRIO sample axiom for the DiVA system.

7. From the OO specification to the CORBA system design

The development cycle of a CORBA application under Iona's Orbix [11] consists of:

- definition of the IDL interfaces for CORBA objects' classes;
- implementation of classes;
- CORBA server program implementation;
- CORBA client program implementation.

The CORBA server program is responsible for making CORBA server objects available through the ORB. The minimum the server program has to do is to instantiate static CORBA server objects and register them under the broker. The client program is responsible for the instantiation of client objects.

In the design of the software architecture of a CORBA implementation of a system specified with UML, the class diagram allows an initial definition of the actual CORBA objects and, for each of them, of the public parts (attributes and operations) to be included in the IDL interface. IDL is the objects' Interface Description Language of CORBA. IDL has a C++ like syntax for the definition of the objects' public interfaces, corresponding to the responsibilities drawn in the UML modeling phase.

The IDL code for DiVA-CORBA object follows.

```
// IDL code for the DiVA objects
interfaces
#include "def.idl" // User data types
definitions

// Definition of exceptions generated by
VideoServer
exception NoAccess { string reason; };
exception NoList   { string reason; };
exception NoURL    { string reason; };
exception Unreach  { string reason; };

// VideoServer interface towards
VideoClient
interface VideoServer {
  void login (in string login, in string
password,
       out unsigned long idclient)
          raises (NoAccess);
  void GetList(out seqFilmInf FilmList)
          raises (NoList);
  void GetURL(in unsigned long idclip ,
out FilmInf Film)
          raises (NoURL, Unreach);
};

// VideoServer interface to VideoClient
interface VS_to_FS {
  void BuildGlobalList(in seqRemoteData
LocalList, in URL location)
          raises (NoList);
};

// definition of exceptions generated by
FilmServer
exception NoValues   { string reason; };
exception NoFilmList { string reason; };
exception NoAccessFS { string reason; };

// FilmServer interface to VideoClient
interface FilmServer {
  void login (in unsigned long idclient)
          raises (NoAccessFS);
};

// FilmServer interface towards
VideoServer
interface FS_to_VS {
```

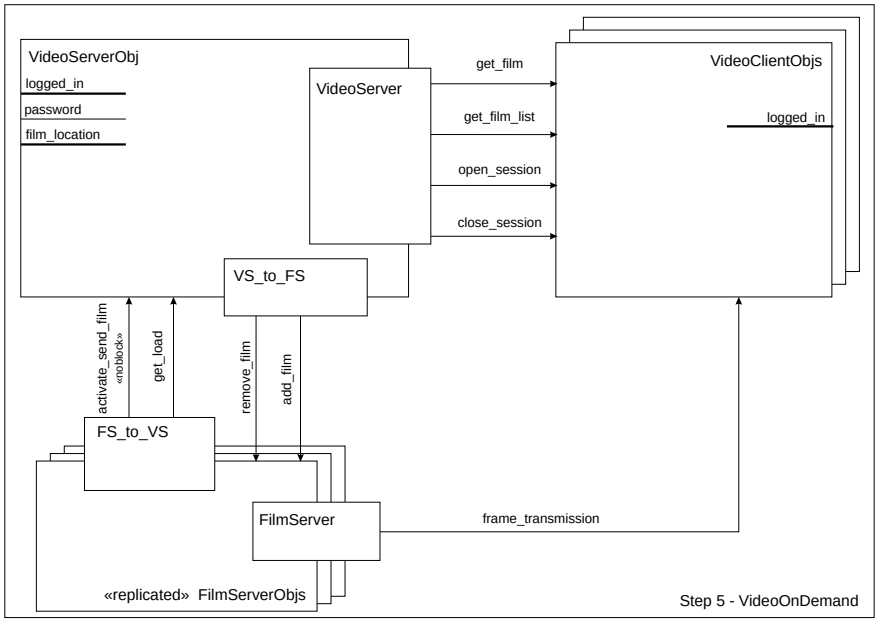

Fig. 8. DiVA class diagram after having applied the T/C methodology.

```
   void GetParamList( inout seqParam
ParamList )
          raises (NoValues);
   void UpArch(in unsigned long idclient);
};
```

The UML component diagram reveals useful to define an initial static allocation of the CORBA objects onto the distributed target platform. Sequence diagrams define the flow of methods' invocation between objects; along with state diagrams, they guide the implementation (coding) and the testing of objects' behaviour (methods).

A drawback of the UML graphical notation is that it lacks some rigor, since its semantics is not precisely defined. On the other hand, this rigor can be provided by the (complementary) use of formal languages.

The TRIO/CORBA language and methodology [12] have been developed to introduce formal languages at architectural design level. Following the TRIO/CORBA (T/C) methodology it is possible to derive the CORBA architecture of an application from its TRIO functional specification. This architecture is rigorously formalized through the T/C language.

The T/C methodology is composed of five steps: 1) identification of data flows between TRIO classes; 2) separation of data flows in operations and attributes and identification of client-server relationships between classes; 3) identification of interfaces and application objects; 4) recognition of non-blocking operations and read only attributes; 5) identification of CORBA services.

'get_film_list') offered by CORBA application objects (in this case of operation 'get_film_list' this is the video server) through IDL interfaces (e.g. 'VideoServer')[3].

The T/C class diagram of figure 8 has a corresponding textual description (expressed in the T/C language), which defines rigorously the behavior (i.e. the semantics) of the application objects and clients that compose the DiVA-CORBA system.

It must be noted that the UML and the T/C approaches are complementary. In fact, UML diagrams can be formalized through the T/C language; viceversa, form the T/C description of an application it is possible to build the corresponding UML diagrams.

The overall architecture of the DiVA-CORBA system is represented in Fig. 9. The graphical user interface (VideoClient) recalls the corresponding CORBA object VideoClient to which it propagates the user requests (message no. 1 in Fig. 9); the requests are sent through to the CORBA AMS object (VideoServer) by means of the ORB (sequence of messages 2.1, 2.2). Replies are then sent by CORBA AMS to CORBA VideoClient (2.3, 2.4). Upon selection of a specific videoclip, CORBA AMS interacts wiht CORBA ArchServer (the public interface of the FilmServer) in order to activate the Sender process that will take care of the frame transmission. Concurrently, a dedicated Receiver process in the CORBA VideoClient is activated, responsible for flow synchronization and display. At this point the real delivery takes place over the channel created between Sender and Receiver.

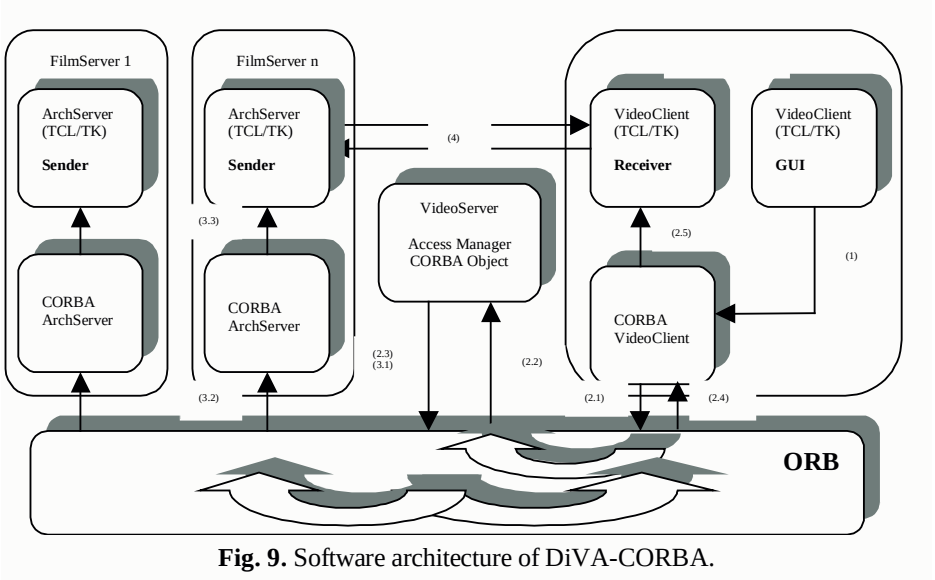

Fig. 9. Software architecture of DiVA-CORBA.

The architecture of the DiVA-CORBA system that we obtained through the T/C methodology is shown in figure 8. All the logic items that in the TRIO class diagram of figure 6 connected two different TRIO classes are now organized in operations (for example

[3] Since the CORBA architecture of the DiVA system was derived from the UML diagrams and from the TRIO specification in two parallel projects, the two CORBA structures can present some discrepancies.

8. Conclusions and future work

We have described the object-oriented specification of a Video-on-Demand system, and discussed advantages of this approach in the development of distributed multimedia applications.

Ease of modeling is among the major benefits of popular OO modeling languages such as UML. When real-time requirements come into play, however, formal specification techniques are required, to support automated checking or at least formal reasoning. UML lacks such features. TRIO combines graphical and textual descriptions; the latter specify behavioural properties as logic formulas.

A pragmatic approach (UML) and a more formal one (T/C) for moving from the OO specification to the definition of the architecture of a CORBA system have been sketched out. Although the VoD case study represents well the class of DMAs, more research work is required in this sense.

Previous work had focused on the use of the PARSE process graph notation to design DMAs [3]. We believe that UML lacks the kind of diagrams that PARSE provides to define communication paths between cooperating objects. These are very useful when building distributed systems where communication is hand-coded based on some message-passing API (like TCP/IP sockets). However, this is not the case of a CORBA implementation, since message-passing is implicit in the semantics of distributed objects' method invocation.

Several other issues still need to be considered. These include the validation that functional and performance requirements are met by the CORBA implementation. Formal validation of the application can be achieved through the TRIO semantic tools [13], once the CORBA platform is also validated. It must be noted that platform validation is one of the goals of the OpenDREAMS ESPRIT project.

Future work on the DiVA case study will address the incorporation of capabilities of fault-tolerance (to recover from user session crashes) and load balancing (among FilmServers). New dedicated objects responsible for managing the status of the distributed system will be added, and the VideoServer will be modified accordingly, to implement predefined policies for managing replicas and session recovery transparently to VideoClients. We shall investigate how such requirements can be specified formally, and then implemented using CORBA services, like the replication service defined in the OpenDREAMS ESPRIT project [14], in such a way that requirements satisfaction can be verified.

Acknowledgements

We acknowledge the contribution of Dr Lucia Merone to the design and implementation of the DiVA system under CORBA.

References

[1] F Fluckiger, *Understanding Networked Multimedia – applications and technologies*, Prentice-Hall (1995)

[6] M. Fowler with K. Scott, *UML distilled*, Addison Wesley (1997)

[3] E.Ciapessoni, A.Coen-Porisini, E.Crivelli, D.Mandrioli, P.Mirandola, A.Morzenti, *From formal models to formally-based methods: an industrial experience*, ACM TOSEM - Transactions On Software Engineering and Methodologies, vol. 8, No 1, January 1999, pages 80-115

[4] R. Orfali, D. Harkey, *The Essential Distributed Objects Survival Guide*, John Wiley & sons (1997)

[5] T J Mowbray, W A Ruh, *Inside CORBA: Distributed Object Standards and Applications*, Addison Wesley (1997)

[6] A Y Liu, T S Chan and I Gorton, *Designing Distributed Multimedia Systems Using PARSE*, Proc IFIP Workshop on Software Engineering for Parallel and Distributed Systems (PDSE96), April 1996, Berlin, Germany, Chapman & Hall (1996)

[7] I Gorton, J P Gray and I E Jelly, *Object Based Modelling of Parallel Programs*, IEEE Parallel and Distributed Technology Journal, Vol 3, No. 2 (1995)

[8] DiVA Project, www.grid.unina.it/projects/diva

[9] Berkeley Continuous Media Toolkit, bmrc.berkeley.edu/projects/cmt/cmt.html

[10] Sullivan et al., Programming with the Java Media Framework, Wiley Computer Publishing (1998)

[11] S. Baker: *CORBA Distributed Objects Using Orbix*. Addison-Wesley – ACM Press (1997)

[12] OpenDREAMS II Consortium, *Intermediate Application Development Methodology*, Deliv. WP5/T5.1-PDM-REP/R51a-V1 (1998)

[13] D. Mandrioli, S. Morasca, A. Morzenti, *Generating test cases for real-time systems from logic specifications*, ACM-TOCS - Transactions on Computer Systems, 13(4) (1995)

[14] OpenDREAMS II Consortium, *Replication Service Design*, Deliv. WP1/T1.4-EPFL-REP/IR14-V1 (1998)