Adaptive QoS Aware Binding of Persistent Multimedia Objects

Thomas Plagemann  
University of Oslo  
Center for Technology at Kjeller  
plageman@unik.no

Frank Eliassen  
University of Oslo  
Department of Informatics  
frank@ifi.uio.no

Vera Goebel  
University of Oslo  
Center for Technology at Kjeller  
goebel@unik.no

Tom Kristensen  
University of Oslo  
Center for Technology at Kjeller  
tomkri@unik.no

Hans Ole Rafaelsen  
University of Tromsø  
Department of Computer Science  
hansr@acm.org

Abstract
The heterogeneity of distributed multimedia systems in terms of hardware, operating systems, programming languages, data formats, compression formats, available resources, and QoS requirements imposes severe problems. The purpose of the middleware platform is to provide interoperability and portability of distributed system services and applications. However, today's middleware platforms, like CORBA and TINA-DPE, do not solve all these problems. For example, automated control of interface compatibility and Quality-of-Service (QoS) support are still important research topics. This is especially true in the context of multimedia database systems (MMDBS) in distributed systems, because interface type and QoS depends on the result of queries and cannot be determined in advance. In this paper, we show how the temporal object-oriented data model TOOMM and the formal model for bindings MBS can be utilized in a heterogeneous distributed system to solve the following problems:

• Determine whether a certain persistent multimedia object and arbitrary clients are computationally compatible; either “as is” or through the transparent introduction of run-time filters, e.g., transcoding from MPEG to H.263.

• Determine whether the Quality-of-Service (QoS) requirements of all communication partners can be adjusted to find a mutually agreeable level.

• Automatically adapt middleware protocols that implement the binding between persistent multimedia objects and clients to the functional and resource related requirements of persistent objects and clients.

We present a vertical approach to automatically combine and adapt applications, persistent multimedia objects, middleware protocols, and transmission facilities. To the best of our knowledge, there is no work reported that solves the problem of interoperability and adaptability in these layers in a single framework. However, there is a considerable amount of related work for each of the three core elements, i.e., TOOMM, MBS, and the adaptable multimedia object request broker (ORB) MULTE-ORB [6]. Due to space limitations, we kindly request the interested reader to refer to the corresponding publications for a discussion of this type of related work.

The outline of this paper is as follows: in Section 2, we motivate the presented work in the context of a concrete application scenario. Afterwards, we describe the three central elements that are used to solve the mentioned problems and discuss their inter-relationships. Section 3 - 5 present TOOMM, MBS, and the MULTE-ORB. In Section 6, we use the application scenario to explain the usage of TOOMM, MBS, and MULTE-ORB to determine interoperability, QoS agreement, and optimally adapt the...

1. Introduction

A basic requirement of distributed multimedia applications is the need to communicate multimedia data of various kinds between sources and destinations. A video conferencing application, for example, typically needs to be able to multicast video streams from each participant location, rendering these into a desktop window at every other participant location. In an open distributed system, however, there is no guarantee that components are built using the same technology. This situation will have severe repercussions of, for example, the ability of a multimedia database system (MMDBS) to interoperate with its clients. The problem of heterogeneity in distributed systems has led to the emergence of object-based middleware platforms such as CORBA [8] and TINA-DPE [12]. The purpose of the middleware platform is to provide interoperability and portability of distributed system services and applications.

In this paper, we show how the temporal object-oriented multimedia data model TOOMM [4] and the formal model for bindings MBS [2] can be utilized in a heterogeneous distributed system to solve the following problems:

• Determine whether a certain persistent multimedia object and arbitrary clients are computationally compatible; either “as is” or through the transparent introduction of run-time filters, e.g., transcoding from MPEG to H.263.

• Determine whether the Quality-of-Service (QoS) requirements of all communication partners can be adjusted to find a mutually agreeable level.

• Automatically adapt middleware protocols that implement the binding between persistent multimedia objects and clients to the functional and resource related requirements of persistent objects and clients.

We present a vertical approach to automatically combine and adapt applications, persistent multimedia objects, middleware protocols, and transmission facilities. To the best of our knowledge, there is no work reported that solves the problem of interoperability and adaptability in these layers in a single framework. However, there is a considerable amount of related work for each of the three core elements, i.e., TOOMM, MBS, and the adaptable multimedia object request broker (ORB) MULTE-ORB [6]. Due to space limitations, we kindly request the interested reader to refer to the corresponding publications for a discussion of this type of related work.

The outline of this paper is as follows: in Section 2, we motivate the presented work in the context of a concrete application scenario. Afterwards, we describe the three central elements that are used to solve the mentioned problems and discuss their inter-relationships. Section 3 - 5 present TOOMM, MBS, and the MULTE-ORB. In Section 6, we use the application scenario to explain the usage of TOOMM, MBS, and MULTE-ORB to determine interoperability, QoS agreement, and optimally adapt the...
MULTE-ORB to the described requirements. Concluding remarks are given in Section 7.

2. Application scenario: Lecture-on-Demand

Interactive distance learning (IDL) refers to all types of studies in which students are separated by space and/or time. The electronic classrooms at the University of Oslo overcome separation in space by exchanging digital audio, video, and whiteboard information multiple sites in Norway. Since 1993, the electronic classrooms are regularly used for teaching graduate level courses as well as for research on Quality-of-Service (QoS) support in distributed multimedia systems [11].

Despite the distributed nature of the electronic classrooms, they support synchronous IDL only. Currently, we develop in the OMODIS project [3] an adaptive distributed multimedia system for Lecture-on-Demand (LoD) to additionally support asynchronous IDL. The core of this system is a MMDBS with QoS support [4]. Recorded lectures and additional multimedia information that is relevant for the courses will be stored in the MMDBS. Main goals for the LoD system are:

- **Individual online access**: users can access the MMDBS via various types of transmission networks, like Ethernet, ATM, N-ISDN, and GSM.
- **QoS support**: if possible, user and application requirements will be considered in the MMDBS, operating systems, communication protocols, and network. Users specify their requirements in each query or the system will use a particular user profile as default.
- **Flexible query facilities**: users have several possibilities to find lectures, e.g., via the date of the lecture, a content description, etc.
- **Independence of data elements/streams**: all events and media streams that are captured in the electronic classroom are independently stored to allow independent retrieval and scalable playback of elements.
- **Scalable playback of lectures**: can be enforced via queries and via QoS requirements. It is possible to retrieve and playback a complete lecture or only parts of it. Scaling of video and audio is also supported.
- **Fully synchronized playback**: Information about the temporal relationships between data elements is stored in the MMDBS as metadata.

Collaborative presentation of multimedia objects is also supported, e.g., one user selects via a query a certain multimedia object in the MMDBS and controls the playback of the object from the MMDBS to multiple clients. Collaborative browsing enables teachers to deepen a certain topic by presenting data from the MMDBS. The teacher issues a query including a QoS specification to the MMDBS. The MMDBS retrieves the data and multicasts it over the network to multiple students, each receiving the data on her/his computer (see Figure 1).

![Collaborative browsing](image)

A specific problem that preferably has to be handled by the multimedia middleware is heterogeneity of student equipment and underlying networks. For example, if the supported performance, audio/video formats, and compression schemes vary, the middleware platform must solve this problem through, e.g., transcoding and media stream scaling. Automatic support of this feature requires a computationally tractable foundation for reasoning about compatibility of the capabilities of systems and networks. An important source for information required by this process is the MMDBS, because the type and amount of data to be transmitted depends on the result of the teachers query.

3. TOOMM

The temporal object-oriented multimedia data model TOOMM [4] provides a formal temporal framework for multimedia data types (MMDTs). TOOMM utilizes concepts from multiple multimedia and temporal object models, most important are AMOS, SGML/HyTime, LMDM, T_Chimera, and Tigukat. TOOMM has several advantages compared to other multimedia object models such as a formal structure for representing time in MMDTs, temporal relationships and structured synchronization information. The separation of presentation information from the actual multimedia data in the SGML/HyTime (see Figure 2) model lead us to create a logical data model and a presentation model as two separate modules. The logical data model type hierarchy in TOOMM is structured as a tree classifying different MMDTs according to their temporal characteristics. Moreover, each multimedia object in the logical data model can have many corresponding presentation objects in the presentation model, making it possible to view the data elements in many different ways. Hence, multimedia objects that are used for different purposes need only to be stored once, decreasing redundancy and preserving integrity in the MMDBS.
Many multimedia application level QoS parameters are present both in the object types of the logical data model and in the presentation model of TOOMM. Temporal relationships contain synchronization requirements between multimedia objects and information on deadlines.

![Diagram of Logical Data Model and Presentation Model]

**Figure 2.** Relationships between logical data model and presentation model

TOOMM provides an advanced framework for creating multiple specialized multimedia presentations based on the same multimedia data without the need for replication. The object types in the presentation model correspond to the MMDTs in the logical data model. Each presentation object type provides an easy way of specializing the presentation of a MMDT. Sets of presentation object types can be combined with temporal relationships to form complete and highly specialized multimedia presentations.

### 3.1. Concepts of TOOMM

Objects in TOOMM comprise the properties of traditional object-oriented data models and three different time dimensions: valid time and transaction time and a new time dimension specifically tailored for MMDTs and multimedia presentations called **play time**. This dimension places *logical data units* (LDUs) of multimedia data, such as frames or audio samples, into a temporal structure for multimedia presentation. Furthermore, TOOMM is based on the following two principles:

- **Separation of multimedia data from its presentation specification.** Objects containing multimedia data are instances of object types from the logical data model. Objects instantiated from the presentation model specify how multimedia data should be presented. We differentiate between *atomic presentation objects* (APOs) describing the presentation of individual multimedia objects, and *composite presentation objects* (CPOs) containing collections of presentation objects and metadata, and supporting the correctly synchronized playback of multimedia data.

### 3.2. Logical data model

The logical data model consists of an extensible class hierarchy of the most common MMDTs such as video, audio, animation, music and basic abstract data types (ADTs). In this hierarchy, we differentiate between the three main categories of MMDTs:

1. **Play time independent multimedia data types (PTI_MMDTs),**
2. **Play time dependent multimedia data types (PTD_MMDTs),** and
3. **Components of PTD_MMDTs.**

ADTs that have static appearance during their presentation are said to belong to the PTI_MMDT category. Basic ADTs such as integer, real, boolean, character, long, etc. belong to the PTI_MMDT category. TOOMM also supports PTI_MMDTs with temporal characteristics: each PTI_MMDT can be extended with valid time and transaction time dimensions. Such an extension of time independent ADTs and PTI_MMDTs with temporal characteristics enables TOOMM to model data history and versions.

In contrast to PTI_MMDTs, PTD_MMDTs comprise all types that have dynamic appearance during their presentation. Based on the PTD_MMDTs temporal characteristics, these object types are further classified into either stream or Computer Generated Media (CGM) sub-categories. Stream objects are always related to a discrete time domain and are of periodic nature. Contrary, components of CGM objects are related to continuous time domains and can be presented in an arbitrary manner.

Component object types are classified according to the sub-category of the PTD_MMDTs they belong to. Component object types of stream object types are denoted LDU object types. Component object types of CGM object types are denoted event object types. By placing components on the same level in the MMDT hierarchy as PTI_MMDTs and PTD_MMDTs, we achieve independence and data reuse. The provision of different frame rates in turn enables us to adjust the presentation and the amount of data to be retrieved according to the QoS requirements of the users.

The logical data model comprises in addition to this MMDT hierarchy two further important features:
(1) metadata to describe the contents of multimedia data and
(2) temporal information, e.g., duration needed to present multimedia data, or information about QoS such as resolution, frame rate, and picture quality for video.
PTI_MMDT objects and PTD_MMDT objects can contain references to metadata, i.e., text describing the contents of the multimedia object. For PTD_MMDT objects, two play time timestamps are used to relate the content description of a certain interval, e.g., a scene in a video, to the data of this interval.

3.3. Play time dimension

Valid and transaction time dimensions are not sufficient to model the temporal nature of multimedia objects and their diversity in time granularities. Therefore, we have developed the play time dimension to handle the temporal nature of time dependent data and different time granularities in a media-independent manner. Play time is used in the logical data model and in the presentation model; it can be seen as the glue between the two models.

Components of a stream multimedia object

Components of a CGM multimedia object

Figure 3. Use of play time in streams and CGMs

In the presentation model, play time is used as a means to map different time granularities of various multimedia objects to the global time granularity. In the logical data model, the play time dimension is used to define a temporal order between all components of multimedia objects. Based on this temporal order, we calculate the relative playback times of all components. Streams and CGMs have different temporal characteristics: in streams, each LDU has to be presented for a fixed duration, called LDU_duration, and in CGMs, each event is related to a start and stop time (see Figure 3).

A major benefit of TOOMM is to combine streams and CGMs in a presentation in a uniform and flexible way. This is achieved through mechanisms for management of mixed granularities, and construction of CPOs for handling the presentation information of different MMDTs. The time scale used in the temporal specifications of a CGM is the play time dimension of that CGM. One granule in that time scale corresponds to a chronon.

3.4. Presentation model

The logical data model supports temporal information for single multimedia objects, like LDU_duration, because this (default) temporal information belongs inherently to the data. However, a single multimedia object might be presented in different ways. Therefore, we differentiate between the default temporal information of multimedia objects and temporal information that is used for a particular presentation of the object. Furthermore, temporal relationships between multimedia objects are not included in the logical data model to support unconstrained combination of multimedia objects in presentations. For instance, a particular video sequence might be presented with audio sequences (speech) and subtitles in different languages. In order to promote data reuse, all temporal relationships that are relevant for presentation are part of the presentation model. Different presentations that use the same data can be independently stored from the data, and the data is only stored once.

We differentiate between two object types in the presentation model: atomic presentation object (APO) types and composite presentation object (CPO) types. APOs are the atomic building blocks of a complex multimedia presentation that is specified in a CPO. Each APO specifies the presentation of a part of, or entire single multimedia object. Thus, for each multimedia object type, TOOMM provides one APO type. APOs typically contain information on:

- References to the multimedia data in terms of play time using the global play time dimension that is defined in the corresponding CPO. APOs that refer to a PTD_MMDT object must specify a continuous sequence of LDUs that have to be presented via start and stop time. In this way, the APO can select a part of, or the entire data set of the multimedia object. APOs that refer to a PTI_MMDT object must specify the time when the PTI_MMDT object should be presented.
- QoS specification of the presentation of the multimedia data can differ from the maximal QoS of the stored multimedia data.
- Effects on the multimedia data such as fade in, fade out, change in volume etc.

CPOs specify the structure of complex multimedia presentations. The main elements of a CPO are a set of APOs and a set of temporal relationships among these APOs. Additionally, it contains the definition of a master chronon, termed Master Time Unit Duration (MTU_duration). The MTU_duration sets the granularity...
of the global play time which all the presentation objects must relate to. CPOs typically contain:

- Temporal relationships among multimedia data presentations. This is done through temporal relationship objects (TROs) which connect presentation objects, and specify their mutual temporal dependencies.
- Alternate multimedia data. For instance, if a video sequence has audio sequences in different languages available then the user should be able to choose between them.

4. MBS

The basic programming concepts for multimedia provided by our MULTE-ORB are operational interfaces, signal interfaces, stream interfaces, stream flows, and associated type checking rules. In this Section, we concentrate on stream abstractions. For the stream abstractions, we have developed a generic formal model called MBS (Model of Bindings and Stream Interfaces) [1]. MBS includes compatibility rules ensuring the correctness of binding attempts of stream flow endpoints, and conformance rules expressing conditions for substitutability. An implementation of the model is described in [2].

4.1. Explicit bindings

In MBS, stream bindings are explicit. This supports direct client control of the binding during its lifetime through its control interface. In Figure 4, a binding object connects two interfaces by means of local bindings that associate the interfaces of the objects with the interfaces of the binding object.

![Figure 4. The explicit binding model](image)

A binding type identifies the type of interfaces which can participate in the binding, the roles they play, and the way behavior at the various binding interfaces are linked [7]. For example, a multicast video binding would typically support a producer role and a consumer role. Binding factories are objects that create bindings. A binding factory is associated to a binding type such that by invoking the factory object’s create method, a new binding of the associated type is created.

Through the local bindings the binding object receives and delivers information for binding participants according to the causality and type of the bound interfaces. Type checking is applied when adding a new interface to the binding. If an object with interface α is offered to fulfil a role β, then the type of α must be compatible with the type of β.

4.2. Flow type model

A stream interface consists of a collection of source and/or sink media flows. A continuous media flow is a (finite) sequence of temporally constrained data elements. Stream interfaces have been adopted in Open Distributed Processing [5], TINA-DPE [12], and OMG [8].

A flow type is specified by indicating the media type of the flow such as audio or video, its causality (source or sink), and a set of quality attributes such as rate and resolution. Furthermore, an attribute value is specified as a set of "atomic" values. For example, when a sink flow type specifies a set of different names on the video encoding attribute, it actually declares that it can accept flows where the video can have any of the indicated formats. The following example of an H.261 video flow type features optional playback rates.

```
| type VideoPhone = Flow |
| V:Video[encoding:H.261, rate:2..24]] |
| A:Audio[encoding:PCM, rate:[8000,16000]] |
| (V & A) | A |
```

This specification states that a VideoPhone flow consists of two different element types labeled V and A respectively. Each element type includes a declaration of the generic media type such as Video or Audio, and a specific set of attributes, specifying quality properties of the element type. The expression (V & A) | A denotes a specification of the structural constraints of the flow [2]. The above structural constraint indicates that an instance of a flow can consist of either video or audio elements, or audio elements only. Hence we may think of the specification as modeling an adaptable flow endpoint.

A flow type specification is interpreted as a set of potential flow qualities and flow configuration that can be produced by a source flow endpoint or is acceptable to a sink flow endpoint. The quality interpretation of a flow type is defined as the combination of the interpretation I of each of its element types (for further details see [1]).

The semantics of the subtype relationship is that of set inclusion. We define a flow type M to be a subtype of the flow type N if both the quality and structural interpretation of M is a subset of the corresponding interpretations of N. We derive that M = Flow[A;B] is a strict quality subtype of N = Flow[C;D], denoted M ≤q N, if I(A) ⊆ I(C) and I(B) ⊆ I(D). A relaxed quality subtype may support fewer element types than the supertype such that, for example, M = Flow[A] is a relaxed quality subtype of N = Flow[B;C], denoted M ≤r N, when I(A) ⊆ I(B).

Type checking binding attempts ensures that the source and the sink can at least support one common flow
quality. We refer to this relationship as quality compatibility. Informally, two flow types are strict quality compatible (denoted \(\sim\)) if their respective element types have pair-wise non-empty set intersection of their respective interpretations. We may for example conclude that \(\text{Flow}[A;B] \sim \text{Flow}[C;D]\) if \(\{A\} \cap \{C\} \neq \emptyset\) and \(\{B\} \cap \{D\} \neq \emptyset\). Two flow types \(M\) and \(N\) are relaxed flow quality compatible (denoted \(\ll\)) if a subset of the element descriptors in each of the two flow types, are pair-wise compatible. Thus, if \(\{A\} \cap \{C\} \neq \emptyset\), \(\text{Flow}[A;B] \ll \text{Flow}[C;D]\) even if \(B\) and \(D\) are incompatible, i.e., \(\{B\} \cap \{D\} = \emptyset\). Two flows are structural compatible, denoted \(\precsim\), if their structural interpretations have non-empty intersection. Different variants of the compatibility relationship where some variants are weaker than others, are the following:

i) fully strict compatible, if \(M \ll N\) and \(M \precsim N\).
ii) partially strict compatible, if \(M \ll N\) and \(M \precsim N\).
iii) fully relaxed compatible, if \(M \ll N\) and \(M \precsim N\).
iv) partially relaxed compatible, if \(M \ll N\) and \(M \precsim N\).

4.3. Stream type model

The notion of configuration constraint allows for the specification of individual requirements to the satisfaction of compatibility for stream interfaces in terms of alternative combinations of flows that may be configured in a stream binding. A stream configuration constraint is written as a structural constraint over flow labels.

Two stream interfaces \(S\) and \(T\) are compatible, denoted \(S \precsim T\), if there are legal configurations of \(S\) and \(T\) such that the flows in these configurations are pair-wise compatible.

Conformance rules express conditions for substitutability. Informally, a stream interface type \(T\) conforms to a stream interface type \(S\) if and only if for every flow of \(S\) there is a corresponding flow in \(T\) such that the flow of \(S\) is a subtype of the flow of \(T\).

4.4. Selection of binding type

Applications select binding types and associated binding factories according to their requirements using a trading model. A set of binding factories is located based on a specification of the required properties of the binding.

4.4.1 Specifying binding types. A binding type is defined as a 5-tuple \((T,P,M,\Delta,E)\) where \(T\) denotes a set of role types, \(P\) a set of roles, \(M\) a set role matching requirements (one for each role), \(\Delta\) a set of role causalities, and \(E\) a set of role cardinality requirements.

A role type \(\tau\) is defined as a set of stream interface types, i.e., \(\tau = \{T_r, \ldots, T_n\}\). A role defines binding object roles and is specified as a role name and a role type, \(r : \tau\). For example, a video conference binding type could define the roles talk and listen where the role talk could be of the role type \{videoConfProducer\} (c.f. example above).

Role matching requirements specify for each role the kind of type matching required for the flows of an interface when the interface is offered to fulfil the role.

Role cardinality is a specification of the number of a particular role the binding object can support (Lindsey et al. 1995). It is modeled as a one-to-one, one-to-many, many-to-many mapping between a role and a set of possible bindings for that role.

As in [6], we define three options how roles can be mapped together:

\[
\begin{align*}
&\langle \text{talk},2..10 \rangle, \\
&\langle \text{talk},1..12 \rangle, \\
&\langle \text{talk},1..12,\text{no_max} \rangle.
\end{align*}
\]

4.4.2. Binding type conformance. An application selects a binding type by stating binding type requirements to a trader that compares the requirements to binding type offers. A binding type requirement specification is with the exception of role matching requirements, identical to a definition of a binding type as outlined above, while a binding type offer is simply a binding type specification.

The role matching requirement of a binding type requirement is specified as a triple \(\langle r, m, \sigma \rangle\) where \(\sigma\) indicates whether a stricter role matching requirement than \(m\) is acceptable (\(\sigma=\text{narrow}\)) or not (\(\sigma=\text{no_narrow}\)). For example, strict compatibility is stricter (i.e., logically implies \(\Rightarrow\)) than relaxed compatibility. Selection is based on a conformance
relationship between binding type requirements and binding type offers.

A binding type offer \( B_1 = \langle T_1, P_1, M_1, \Delta_1, E_1 \rangle \) conforms to a binding type requirement \( B_2 = \langle T_2, P_2, M_2, \Delta_2, E_2 \rangle \) if and only if there exists a bijection \( \beta \) between the sets of role causalities \( \Delta_1 \) and \( \Delta_2 \) such that for all \( (\delta, \epsilon) \in \beta \) with \( \delta = \langle C_1, r_1, S_1, m_1 \rangle \) and \( \epsilon = \langle C_2, r_2, S_2, m_2 \rangle \), \( \delta \) satisfies \( \epsilon \) and the role cardinality requirements \( \langle r_1, i_1 \rangle \) and \( \langle S_1, j_1 \rangle \) in \( E_1 \) satisfies the corresponding role cardinality requirements \( \langle r_2, i_2 \rangle \) and \( \langle S_2, j_2 \rangle \) in \( E_2 \) such that \( i_2 \subseteq i_1 \) and \( j_2 \subseteq j_1 \), and the role matching requirements \( \langle r_1, m_1 \rangle \) and \( \langle S_1, n_1 \rangle \) in \( M_1 \) satisfies the corresponding role matching requirements \( \langle r_2, m_2, \sigma_2 \rangle \) and \( \langle S_2, n_2, \mu_2 \rangle \) in \( M_2 \).

A role matching requirement \( \langle r, m, \sigma \rangle \) of a binding type offer satisfies a role matching requirement \( \langle r, m, \sigma \rangle \) of a binding type requirement if and only if \( m \equiv m, \sigma = \text{narrow} \) and \( m \Rightarrow m \). A role causality \( \langle C_1, r_1, r_2, m \rangle \) is satisfied by a role causality \( \langle C', S_1, S_2, n \rangle \) if and only if \( C = C' \), \( S \) conforms to \( r_1, S_1 \) conforms to \( r_2, S_2 \) and if \( n \neq \mu \), then \( n = \text{conv} \). A role type \( \tau = \{ T_1, \ldots, T_i \} \) conforms to a role type \( \sigma = \{ S_1, \ldots, S_i \} \) if and only if there exists a bijection \( \beta \) between \( \tau \) and \( \sigma \) such that for all \( (T_i, S_j) \in \beta \), \( T_i \) conforms to \( S_j \).

Example. Suppose the following binding type requirement of an audio conference application:

```plaintext
talk : audioConfProducer -- roles
listen : audioConfConsumer
```

```plaintext
type audioTalk = stream [ -- stream interfaces
  a : sink flow [  
    al: Audio [encoding: {PCMA, GSM},  
                  rate: 8000] ];  
  configurations a ] //end stream
```

```plaintext
type audioListen = stream [  
  a : source flow [  
    al: Audio [encoding: {PCMA, GSM},  
                rate: 8000] ];  
  configurations a ] //end stream
```

4.5. From presentation objects to stream interfaces

When the result of a query is a composite presentation object (CPO), the database needs to create the appropriate operation, signal and stream bindings to convey the stream and CGM data to the client(s). This requires that the stream components of the CPO (i.e., the APOs) are mapped to appropriate concepts of MBS, and CGM components are mapped to operational or signal interfaces. Below, we concentrate on the mapping of stream components.

In general, the stream mapping can be done in a number of ways. Therefore, we assume that the mapping is supported by a mapping policy defining how each concept of TOOMM should be mapped to MBS. Table 1 summarizes an example of a mapping policy. For completeness, it also includes the mapping of CGM components, although the details of this are beyond the scope of this paper.

Under this mapping policy, for example, a video APO is mapped to a flow with a single video element type with rate parameter value corresponding to LDU_duration, and appropriate encoding attribute value. If the database can support alternative playback rates and/or alternative encodings, this is also reflected in the element type.
specification as set-valued attributes. Furthermore, the
collection of stream category APOs of a composite
presentation object (CPO) is mapped to a stream interface
that contains a set of source flows each corresponding to
one of the APOs of the CPO.

Table 1. Mapping between TOOMM and MBS

<table>
<thead>
<tr>
<th>TOOMM</th>
<th>MBS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stream APO</td>
<td>Flow type</td>
</tr>
<tr>
<td>_DU</td>
<td>Flow Element</td>
</tr>
<tr>
<td>Stream APOs of a CPO</td>
<td>Stream interface (role)</td>
</tr>
<tr>
<td>CGM APO</td>
<td>Signal type</td>
</tr>
<tr>
<td>CGM APOs of a CPO</td>
<td>Signal interface (role)</td>
</tr>
<tr>
<td>Event</td>
<td>Signal parameter</td>
</tr>
</tbody>
</table>

5. MULTE-ORB

In the MULTE (Multimedia Middleware for Low-
Latency High-Throughput Requirements) project, we
develop an adaptive multimedia ORB, called MULTE-
ORB, that is based on a flexible protocol framework. A
flexible protocol system allows dynamic selection,
configuration and reconfiguration of protocol modules to
dynamically shape the functionality of a protocol to satisfy
specific application requirements and/or adapt to changing
service properties of the underlying network. The basic
idea of flexible end-to-end protocols is that they are
configured to include only the necessary functionality
required to satisfy the application for the particular
connection. This might even include filter modules to
resolve incompatibilities among stream flow endpoints
and/or to scale stream flows due to different network
technologies in intermediate networks. The goal of a
particular configuration of protocol modules is to support
the required QoS for requested connections. This will
include point-to-point, point-to-multipoint, and multipoint
–to-multipoint connections. As a starting point, we use the
Da CaPo (Dynamic Configuration of Protocols) system
[9] to build the MULTE-ORB [6].

5.1. Overview of Da CaPo

Da CaPo splits communication systems into three
layers denoted A, C, and T. End-systems communicate via
the transport infrastructure (layer T), representing the
available communication infrastructure with end-to-end
connectivity. In other words, T services are generic and
could be the service of IP in the Internet, a native ATM
service in an ATM network, or the medium access layer in
a local area network. In layer C, the end-to-end
communication support adds functionality to T services
such that at the AC-interface, services are provided to run
distributed applications (layer A). Layer C is decomposed
into protocol functions instead of sublayers. Each protocol
function encapsulates a typical protocol task like error
detection, acknowledgment, flow control, encryption/
decryption, etc. Data dependencies between protocol
functions are specified in a protocol graph. T layer
modules and A layer modules terminate the module graph
of a module configuration. T modules realize access
points to T services and A modules realize access points
to layer C services. Both module types "consume" or
"produce" packets. For example, in a distributed video
application a frame grabber and compression board
produces video data.
include determining appropriate protocol configurations and QoS at runtime, ensuring through peer negotiations that communicating peers use the same protocol for a layer C connection, initiates connection establishment and release, and handles errors which cannot be treated inside single modules. Furthermore, Da CaPo coordinates the reconfiguration of a protocol if the application requirements are no longer fulfilled. Figure 5-a illustrates the decreasing levels of abstraction of protocol functions, protocol mechanisms that can be seen as specifications, and their implementations called modules. A sample protocol graph is illustrated in Figure 5-b and one possible realization of the abstract protocol functions as implementation modules. The protocol function “error handling” is instantiated with the empty module, that means error handling is not performed.

5.2. Integration of Da CaPo in COOL

Currently, we develop a new multithreaded version of Da CaPo on top of the real-time micro-kernel operating system Chorus to take full advantage of the real-time support of Chorus [10]. Furthermore, we integrate Da CaPo into the CORBA implementation COOL such that the MULTE-ORB is able to negotiate QoS and utilizes optimized protocol configurations instead of TCP/IP. Figure 6 illustrates the architecture of the MULTE-ORB on top of Chorus.

The COOL communication subsystem is split into two parts to separate the message protocol, i.e., the Inter-ORB Protocol (IIOP) and the proprietary COOL Protocol, from the underlying transport protocols, i.e., TCP/IP and Chorus Inter-Process Communication (IPC). A generic message protocol provides a common interface upwards, thus, generated IDL stubs and skeletons are protocol independent. A generic transport protocol provides a common interface for the different transport implementations.

There are two alternatives to integrate Da CaPo in this architecture: (1) Da CaPo represents simply another transport protocol, and (2) Da CaPo replaces transport and message protocol. The first alternative is our current prototype implementation for Da CaPo in COOL, accompanied with an extended version of IIOP called QoS-IIOP, or QIOP. QIOP encapsulates QoS information from application level IDL interfaces and conveys this information down to the transport layer and performs at the peer system the reverse operation. Da CaPo uses this information for configuration of protocols. This prototype is able to implement binding objects that support QoS and real-time requirements.

The next step in our work is to implement the second alternative, where Da CaPo additionally configures a message protocol. The message protocols are then Da CaPo modules formatting requests for marshaling and demarshaling in stubs and skeletons. This alternative is also the basis for a new stream adapter in the MULTE-ORB.

5.3. Relationships between MBS and MULTE-ORB

Protocol graphs are abstract descriptions of the functionality of a service and can be directly used to describe a binding type. By instantiating each protocol function with a module, we create a module graph that implements a binding object. The combination of a module graph, the runtime environment, and the signaling protocol corresponds to a binding factory. Table 2 relates the most important concepts in MBS and MULTE-ORB.

<table>
<thead>
<tr>
<th>MBS</th>
<th>MULTE-ORB</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Instance of) binding</td>
<td>Instantiated protocol configuration</td>
</tr>
<tr>
<td>object</td>
<td>(module graph)</td>
</tr>
<tr>
<td>Binding factory</td>
<td>Runtime environment, signaling</td>
</tr>
<tr>
<td>Binding type</td>
<td>protocol, and module graph</td>
</tr>
<tr>
<td>Role</td>
<td>Sending/receiving peer</td>
</tr>
<tr>
<td>Interfaces</td>
<td>A modules</td>
</tr>
</tbody>
</table>

6. Example

In this Section, we discuss an example scenario in the LoD application to illustrate the usage of the previously introduced concepts. A teacher submits a query to the MMDDBMS to select a part of a previous lecture that should be presented to a number of students. The teacher starts the presentation and during the ongoing presentation additional students join the presentation. In this scenario, we identify the following important steps:
1) The teacher submits the query "SELECT transparencies, light-pen, video, audio FROM MOS-lectures WHERE date = 19/2/1998 AND topic = files OR topic = directories" to the MMDBS to repeat the discussion of files and directories from the lecture given at February 19, 1998.

2) The MMDBS processes the query and returns an object identifier (OID) of the resulting CPO to the teacher.

3) The teacher starts the presentation. In order to transmit the data from MMDBS to all clients, i.e., teacher and students, it is necessary to create appropriate binding objects between them. We concentrate on the stream data. The stream binding object is created in three steps:
   a) The MMDBS determines the requirements to the stream binding type to be used for transmitting the stream data to the clients and selects a binding factory that can create a conforming binding.
   b) The binding factory determines compatibility between the student and teacher interfaces and the corresponding roles of the binding, and selects a binding template (protocol graph and module graph) for this binding.
   c) The MULTE-ORB instantiates the corresponding protocols, establishes the binding and transports the multimedia data to the clients.

6.1. Modeling and implementation of a lecture

The logical data model of TOOMM can be used in the following way to support all MMDTs that are part of a lecture:
- **Audio and video**: The Audio and Video MMDTs can be used directly to model the audio and video streams in the electronic classroom. The presentation object type of audio P_Audio and video P_Video can also be used directly.
- **Document camera and scanner**: The data coming from these two devices can be stored as a picture using the Picture MMDT.
- **HTML documents (electronic whiteboard)**: The presentation object types of the relevant MMDTs must recognize the electronic whiteboard as an output unit. This can be achieved by sub-classing the appropriate presentation object type.
- **Metadata**: The data types needed to model metadata about courses are only basic data types such as numbers and text. These basic data types can easily be modeled and associated with the complex multimedia objects.
- **Light-pen and other input devices**: To model the light-pen, the keyboard and the mouse, we must find out where the input units fit into the logical data model type hierarchy. Since the actions from the input units happen at irregular intervals with random duration, the CGM MMDT matches the temporal characteristics most accurately.

Components are time-stamped individually relative to the start-time of the recording of a stream or CGM, not the global start-time. For every event happening during a lecture, the capturing application must create one MMDT object from the logical data model and one P_MMDT object from the presentation model. The MMDT objects keep track of their own temporal information while the P_MMDT objects relate their corresponding MMDT object to the entire lecture.

The CPO contains APOs for the corresponding LDUs satisfying the query. Thus, the CPO contains most of the information that is required by the MBS. Additional information, like data format and compression format, can be retrieved from the meta-data in the data dictionary of the MMDBS.

6.2. Using MBS to analyze interoperability and negotiate properties of the binding

MBS is used to select an appropriate binding for the lecture presentation. This requires that a corresponding binding type requirement specification is generated by the database system. The foundation for this is partly the QoS requirements associated with the query, and partly the properties of the CPO to be presented. The QoS requirements of the query specify the capabilities of each receiver of the presentation in terms of supported audio/video formats and qualities, and in terms of required configurations of streams and CGMs. For example, one of the students wants to receive audio only, because she is blind and another student wants to receive video, transparencies, and light-pen events only, because she is deaf.

First, the database system constructs a stream interface and signal interface specification from information associated with the CPO and additional information from the data dictionary. This is done according to a mapping policy, in this case the one outlined in Section 4.5. Hence, the generated stream interface specification consists of a video source flow type and an audio source flow type, while the signal interface consists of one out signal type for slide events and one out signal type for light-pen events. A role type AVtalk is associated to the stream interface type, and a role type CGMtalk to the signal
interface. Also, a role talk of type AVtalk and a role send of type CGMtalk are defined.

A stream interface and signal interface specification of the receivers are constructed from QoS information associated with the query. The stream interface specification must capture all different capabilities of the teacher and all the students such as all variants of encoding and quality requirements. This is achieved by “unifying” different requirements in corresponding set-valued attributes. The configuration constraint of the stream interface must specify audio only and video only as legal stream configurations (c.f. requirement above). Lastly, a corresponding role listen of type AVlisten is defined. Similar considerations apply to the signal interface resulting in a role recv of type CGMrecv being defined.

From the above role specifications, a binding type requirement can be constructed. In particular role causalities, role matching, and role cardinality requirements are defined. For example, for the stream binding the required role causality is <ONE-MANY,talk,listen,conv>. Conversion is required because of heterogeneous clients.

Using this specification of binding type requirement, an appropriate binding factory can now be located. The binding factory determines the actual behavior to be used at each local binding based on the compatibility relationship. Negotiation is required in those cases where alternative behaviors are possible.

In the resulting binding configuration audio and video are distributed via a stream binding, and transparencies and light-pen events via a signal binding (see Figure 7). Student 3 receives audio only, and student 1 receives video, transparencies, and light-pen events only.

6.3. Tailoring the MULTE-ORB

The task of the binding factory is to select module graphs for each site involved in the binding and to instruct the MULTE-ORB to instantiate the binding. In general, binding factories can be constructed in a number of ways.

At one extreme there are binding factories with hard-coded module graphs offering no flexibility. In our case, however, the binding needs to be tailored to its negotiated properties at each interface. One way to achieve this is to statically associate a protocol graph to the binding type. The protocol graph includes all functionality potentially needed to support the binding. The actual functionality needed depends (among others) on the negotiated properties of each local binding. For example, transcoding is potentially needed between all pairs of causally related role instances. However, if the same encoding is negotiated at related end-points, a transcoder module will not be needed in this instance of the binding.

In the MULTE-ORB, Da CaPo instantiates at each site the selected module graph (see Figure 8). In the capsule of the MMDBS, we do not need an MPEG module because the data is stored in this format. However, all clients require an MPEG decoder, except client 3, because it only presents audio. Due to the fact that the clients of student 1 and student 2 do not have an MPEG decoder, but an H.263 decoder, we need a transcoder MPEG-to-H.263 within the network. The role listen defines for the client that we have to take receiving modules to implement the corresponding protocol entity. Da CaPo’s connection management component coordinates instantiation and initialization of the modules at all clients. If these tasks are successfully performed, the binding object is established.
7. Conclusions

To solve the problems imposed by heterogeneity in distributed systems is a great challenge. This is especially true for distributed multimedia systems, where hardware, operating systems, programming languages, data formats, compression formats, available resources, and QoS requirements are diverse. Today’s object-oriented middleware platforms like CORBA do not sufficiently solve all these problems. In particular, automatic determination of compatibility between application interfaces, QoS support, and adaptation of middleware to the particular application needs require new solutions. In the context of object-oriented MMDBS, this problem is especially complex, because the interface of the MMDBS cannot be specified in advance, it is depending on the outcome of a query (e.g., the data format). However, the MMDBS contains information about the type of the query result.

In this paper, we have shown how the data model TOOMM can be used to derive at runtime information that enables the middleware to reason about interoperability, compatibility and appropriate middleware configurations. Furthermore, we have shown how these results can be used to optimally tailor the MULT-E-ORB to application requirements. Thus, data model and metadata can be used to tailor middleware, down to the basic network facility, e.g., resource reservation with RSVP.

Currently, all major elements, i.e., TOOMM, MBS, and MULT-E-ORB are implemented. The integration of these elements is under development. There are still open problems to solve, but already our intermediate results demonstrate the potential of our work:

- The MMDBS can provide sufficient information to MBS to resolve interoperability, compatibility, and QoS agreements.
- Additionally, MBS is able to select the appropriate binding factory that can generate a binding with properties as required by the application, i.e., DB clients and server.
- The MULT-E-ORB can configure at runtime optimal middleware configurations.

We believe that the vertical approach of our work to automatically combine and adapt applications, persistent multimedia objects, middleware protocols, and transmission facilities represents a significant contribution for the design of future distributed multimedia systems.

8. References


