A UML approach to the design of Open Distributed Systems

Behzad Bordbar, John Derrick, Gill Waters
Computing Laboratory, University of Kent, Canterbury, CT2 7NF, UK.
(Email: B.Bordbar@ukc.ac.uk.)

May 18, 2002

Abstract

The design of distributed systems is a highly complicated and non-trivial task. Introduction of multiple types of media into distributed systems causes a dramatic increase in the complexity of design. To deal with the inherent complexity of systems, two approaches have received considerable attention; ODP and UML. Open Distributed Processing (ODP) is a joint ITU/ISO standardisation framework for constructing distributed systems. Unified Modelling Language (UML) is a de facto standard for visualising, specifying, designing, and documenting object-oriented systems.

This paper presents a case study using a UML approach for the design and specification of distributed systems based on ODP. The purpose of the case study is to try this approach on a large system containing multiple types of media. The case study is carried out on an Interactive Multimedia Kiosk (IMK) example. IMKs integrate different types of media such as text, graphics, audio, video, animation and sound in the form of a large system; this provides an ideal subject for case study.

Keywords: Open Distributed Processing; COMET; Computational Viewpoint; Quality of Service; Specification; UML; Object Constraint Language.

1 Introduction

There have been dramatic recent developments in the area of distributed systems fuelled by the birth of new applications in our daily life, such as different web-based products and the introduction of multimedia systems into day-to-day amenities such as shopping centres, transport facilities and leisure establishments. However, the design of distributed systems is a highly non-trivial task. In particular, introduction of different media types into systems increases the complexity of the design and results in the creation of a new set of problems such as performance related Quality of Service (QoS), portability and interoperability.

To ease the burden of dealing with the complexity of distributed systems, two approaches have received considerable attention; ODP and UML. The Reference Model for Open Distributed Processing (RM-ODP) [7] is a joint ITU/ISO standardisation framework for constructing distributed systems in a multi-vendor environment. Significant features of ODP include *object based* specification and programming, use of *transparencies* to hide aspects of distribution and its use of *viewpoints*. ODP viewpoints are used to help partition the complexity in distributed systems design. Each viewpoint considers the specification of a distributed system from a particular perspective, and of particular relevance to QoS specification is the *computational viewpoint*. This

viewpoint is concerned with the algorithms and data flows which provide the distributed system function. It represents the system and its environment in terms of objects which interact by transfer of information via interfaces.

Since modern distributed systems are object-based, there has been considerable interest in the use of the Unified Modelling Language (UML) [15]. Indeed, UML has quickly emerged as the standard object-oriented analysis and design notation. However, because it attempts to provide notation for most aspects of object-oriented design, users can select from a rich variety of design diagrams in UML and there are few guidelines on how precisely the design is defined.

The RM-ODP presents a framework for constructing distributed systems and defines viewpoints with associated viewpoint languages. However, ODP does not prescribe a specification or design notation within those viewpoints. Thus, there is a need for a specification and modelling language to instantiate the viewpoints and UML is as an ideal candidate for such a language. Various approaches to integrate UML and ODP have been described in [14, 11, 13, 1, 2]. In [2], we presented a UML based method for modelling distributed systems in a framework complying with the RM-ODP. To address the performance issues, in [2] we elaborate on specification of the QoS aspects of the system as a part of the attributes of the entities of the system.

One major objective of introducing a design method for distributed systems is to cope with large examples involving different types of media in sufficient details to enable evaluation of the system in terms of performance, consistency, etc. In this paper we illustrate our approach via specification of static aspects of *Interactive Multimedia Kiosks* (IMKs). We also model the IMK via a UML based but non-ODP approach, COMET [6] and compare the result. We also compare UML and ODP and explain the contribution of our approach to bridging the gap between them.

The paper is organised as follows. The next section introduces IMKs and introduces our running example. Section 3 and 4 uses our running example to provide a brief introduction to ODP and reviews the design method of [2]. Section 5 applies the design method to the IMKs. We shall make a comparison between RM-ODP, UML and the method of [2] in section 6 by pointing out some of the concepts of RM-ODP that are missing from UML and the way that they are introduced into our UML model. Section 6 also presents a model of IMK via COMET [6] to compare it with our method. Finally, we conclude in section 7.

2 Interactive Multimedia Kiosk

An *Interactive Multimedia Kiosk* (IMK) [5] is a public stand that supplies text, graphics, video, animation and sound information to the user. Our example Museum Information Kiosk (MIK) is a real-time coordination of display of information related to a museum offering visitors interactive input, which is based upon one used by Blair and Stefani in [3]. Its function can be divided into three phases as follows.
The welcome phase: The visitor is presented with a welcoming video sequence with associated audio explanation. This phase is terminated when the user presses the start key.
The menu phase: A menu is displayed and a short video and audio message is played repeatedly, which invites the user to select a presentation. This phase is terminated when a user selects a choice from the keyboard.
The conference phase: A sequence of N images accompanied by an associated audio commentary is presented. The exact location of the presented work in the museum building is displayed in a separate window. The audio commentary is divided into N subunits and presented in synchronisation with the corresponding image. The next image is presented when the audio commentary of the previous image has finished. The new audio commentary starts two seconds

after the display of the image. In addition, the access map is displayed simultaneously with the first picture of the conference. On termination of the presentation, the application returns to the menu phase.

The user can temporarily suspend and resume the conference phase with a toggle key with values *suspend* and *resume*. The duration of such a suspension is limited to 30 seconds. At the end of the 30 seconds if the user has not pressed *resume*, the presentation resumes automatically. It is possible to leave the menu or conference phases by pressing a *kill* button, which transfers the system to the welcome phase. There is a timeout of 5 minutes in the menu phase within which if the user has not made a choice, the system assumes that he/she has left and returns to the welcome phase.

3 Open Distributed Processing

The Reference Model for Open Distributed Processing (RM-ODP) [7] describes an architecture for building open distributed systems [10] in a multi-vendor environment. Central to the RM-ODP is the concept of *viewpoints*. Viewpoints partition a system specification into a number of partial descriptions, each targeted towards a particular audience to avoid the wide scope and inherent complexity of the domain. The reference model defines five viewpoints: *enterprise*, *information*, *computational*, *engineering* and *technology*. The following subsection explains the computational viewpoint, which is the focus of this paper. Further information regarding both the reference model and its approach to using viewpoints can be found in [7, 10, 12].

3.1 ODP Computational viewpoint of the IMK

The computational viewpoint deals with the logical partitioning of the distributed system into a series of interacting entities, which are referred to as *objects*. To avoid confusion with the word "object", which is also a reserved word in UML, we use the term computational object, or *compobj* for short. Fig. 1 depicts a high level description of the computational viewpoint of the MIK system of Section 2.

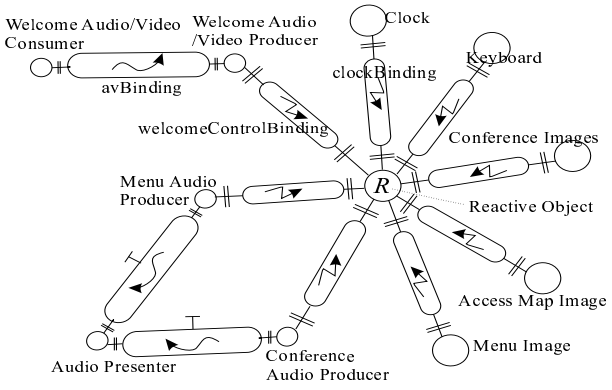

Figure 1: Museum Information Kiosk

In Fig. 1, empty circles depict objects; are ten compobj as follows. *Welcome Audio/Video producer* produces audio and video frames which must be presented via the *Welcome Audio/Video consumer* while the system is in the welcome phase. The audio producer of the phase Menu and

Conference are modelled via compobjs *Menu audio producer* and *Conference audio producer*. *Audio Presenter* models the compobj that consumes the audio, for example, a speaker broadcasting sound. There is a *Keyboard* compobj for entering data and an external *Clock* compobj. Finally, there are three image sources modelled as compobjs *Conference images*, *Access map image* and *Menu image*, which refer to the corresponding repositories of images.

To perform a service in a distributed environment, the computational objects involved need to access one another, through (possibly multiple) *interfaces*. Such interfaces partition the external behaviour associated with computational objects into logically distinct categories enabling a computational object to interact with more than one other object. An interface is known to its environment by its *interface reference*. For example, consider Fig. 2, which depicts the subsystem of Fig. 1 dealing with welcome phase. The compobj *Welcome Audio/ Video Producer* has two interfaces *WelcomeAvControl* and *WelcomeAvOut*. Each interface of a compobj is depicted via a "T" shape attached to it.

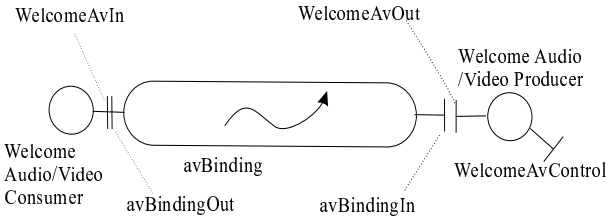

Figure 2: Audio/Video subsystem for the welcome phase

The computational viewpoint defines three types of interfaces: *operational*, *stream* and *signal*. Operational interfaces support invocation of operations on potentially remote computational objects and may be *interrogations* or *announcements*. A stream interface, for example *WelcomeAvOut*, represents a continuous flow.

A signal is an atomic action resulting in a one-way communication from an *initiating object* to a *responding object*. The interface *WelcomeAvControl* of Fig. 2 is an example of a signal interface.

In order for interactions between objects to occur, the interfaces of the relevant objects are associated by the formation of a *binding*, which can be implicit or explicit. In an implicit there is no term for expressing the binding action. In an *explicit binding* the binding itself is encapsulated as an object which provides the infrastructure resources supporting the communication [3, 7]. We shall refer to such objects as *bindobj* to distinguish them from compobjs. For example, the binding between the producer and consumer in the welcome phase of Fig. 2 is modelled by a bindobj called *avBindin*. Bindobjs, like compobjs, interact with the environment via interfaces. The bindobj *avBinding* has two stream interfaces: *avBindingIn* receives the audio/video frames from the environment; following the binding action, frames are delivered on the interface *avBindingOut*.

To the model real-time controllers that offer synchronisation *reactive objects* [3] are introduced. A reactive object interacts with its environment exclusively through signal interfaces. For example, in the multimedia presentation of Fig. 1 in order to synchronise and control the behaviour of the interacting components a reactive object is included. A reactive object is depicted by a circle containing letter "R".

4 A UML approach to computational viewpoint modelling

In this section we shall briefly outline our modelling approach of [2], which provides a guideline for modelling the computational viewpoint of ODP via UML diagrams. First we shall explain how our approach is inspired by and relates to ODP and UML.

The static architecture of the UML is based on a four layered structure of *user object*, *model*, *metamodel* and *metametamodel* [15]. The metametamodel defines the language for specifying metamodel structure. A metamodel, which is an instance of the metametamodel, defines the language for specifying the model. Models, which are instances of the metamodel, define the language used to describe an information domain from which user objects, describing a specific application domain, are specified. As a result, each layer defines a method of specification of the layer below. In this paper we shall only be dealing with the bottom three layers.

Similarly, a three layered structure for ODP can be considered. One can think of the "Computational viewpoint of RM-ODP" as a top layer which describes the computational aspects of the architecture for building open distributed systems. Such descriptions result in a second layer which is a template specifying the Specific Domain of Application (SDoA). The SDoA models the computational aspects of applications with some common features. For each application that belongs to the SDoA the "computational specification of an application " can be created with the help of the corresponding template.

Inspired by the above three layer structure, [2] models the computational viewpoint as a UML diagram, which we call the Computational Metamodel Diagram (CMD). Different computational entities and their relationship are modelled in the CMD. For each Specific Domain of Application (SDoA), for example IMKs, [2] presents a heuristic for modifying the CMD and creating a class diagram which models the static aspects of the system. We shall refer to such class diagrams as the Computational Class Diagrams (CCD). As a result, the CCD models the "Computational Specification of a SDoA". The CCD serves as a template for modelling static aspects of the computational viewpoint of applications which belong to the Specific Domain of Application (SDoA). As a result, by applying a heuristic to CCD, for each such application, an object diagram called the *Computational Object Diagram* (COD) is created which models the static aspects of the computational viewpoint of such application.

4.1 Computational Metamodel Diagram

Fig. 3 is the Computational Metamodel Diagram (CMD), which models the computational viewpoint of RM-ODP via a UML class diagram.

The CMD consists of a number of classes modelling different entities of the computational viewpoint. For example, the class *SystemComponent* embodies classes of computational objects and binding objects. Similarly, the class *ReactiveObject* represents the class of all reactive objects. The class *Infs* represents interfaces to computational or binding objects, and the class *RSigInfs* represents interfaces to the class of all reactive objects which is always a signal interface.

Between the classes in the CMD there are a number of *associations*, and each association models the relationship between the classes that it connects. For example, the association between *SystemComponent* and *Infs* models interfaces assigned to computational or binding objects. Similarly, the association between *ReactiveObject* and *RSigInfs* models the signal interfaces of the reactive objects.

As an example, we shall explain the class *SystemComponent* which has five attributes. *Name*

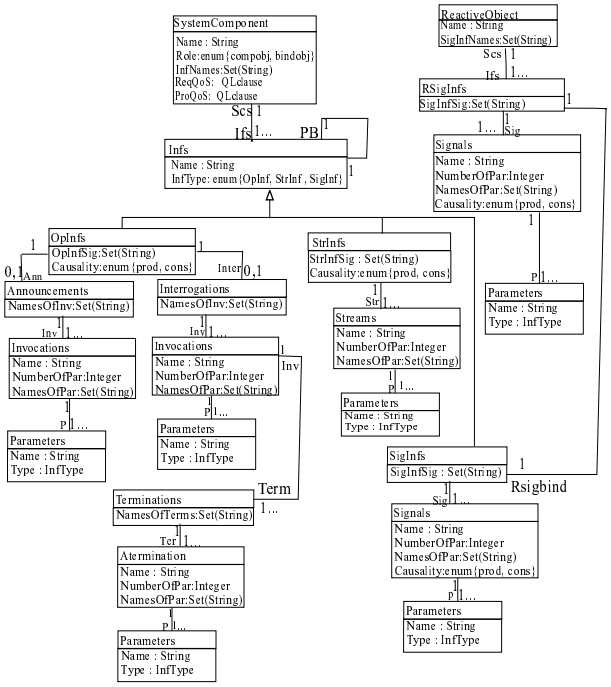

Figure 3: The Computational Metamodel Diagram

identifies the *SystemComponent*. *Role* is used to differentiate between computational objects and binding objects, and has type *enumeration* (*enum*) which lists the set of possible values. The names of the interfaces of each computational or binding object are listed by the attribute *InfNames*.

Attributes *ReqQos* and *ProQoS* of the class *SystemComponent* model *required* and *provided* QoS, respectively. The QoS *provided* to the environment by the computational object and the QoS *required* by the object from its environment [3]. A computational object guarantees to supply the provided QoS to the environment only if it receives the required QoS from the environment and is referred to as a *contractual approach* to QoS. We specify each of these attributes via QL [3], a first order real-time logic based on RTL [8]. As a result, the type of such attributes is specified as QL clauses (*QLclause*).

The class *Infs* models the interfaces in ODP and has two attributes of *Names* and *InfType* which refer to the name and the type of the each interface. Types of interfaces are operational interfaces *OpInf*, signal interfaces *SigInf* or stream interfaces *StrInf*. Different types of interfaces are modelled via three sub-classes *OpInfs*, *StrInfs* and *SigInfs* corresponding to operational interfaces, stream interfaces and signal interfaces in the computational model, respectively. Thus, in UML terminology, the class *Infs* is a generalisation of classes *OpInfs*, *StrInfs* and *SigInfs*.

5 IMK as a Specific Domain of Application

In this section, we focus on an Interactive Multimedia Kiosk (IMK) as a Specific Domain of Application (SDoA) and use the CMD of Fig. 3 to produce its Computational Class Diagram. The CCD acts as a template for instantiation of object diagram of different applications such as

the MIK belonging to the SDoA IMK.

5.1 Computational Class Diagram for IMK Systems

A heuristic for the creation of Computational Class Diagrams (CCD) from the CMD of Fig. 3 is explained in [2]. To avoid confusion, for the rest of the section the word *metaclass* refers

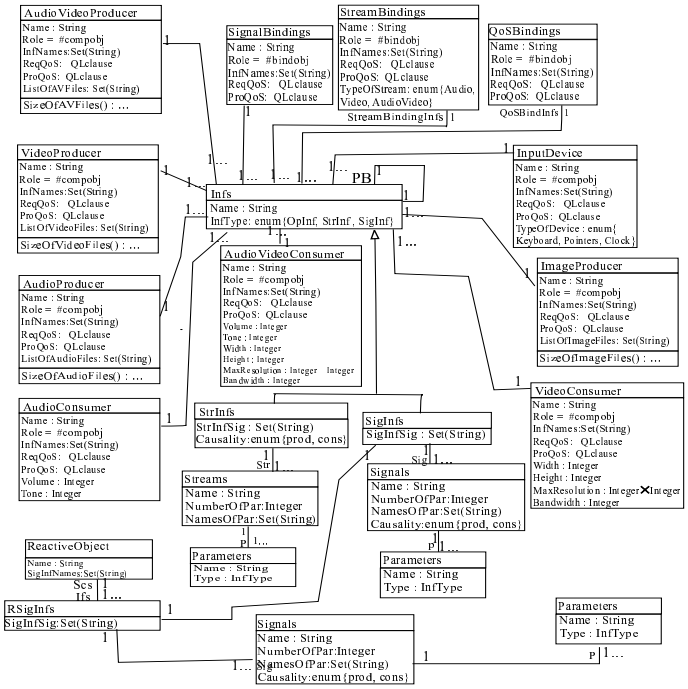

Figure 4: The CCD of the Interactive Multimedia Kiosk

to the CMD of Fig. 3 which acts as a template for creation of Fig. 4. First, we start from metaclass *SystemComponent* and create classes *AudioProducer*, *VideoProducer*, *ImageProducer* and *AudioVideoProducer* which model the resources which are manipulated to produce sounds, graphics, pictures and video clips files. For example, the class *AudioProducer* which is created from the template metaclass *SystemComponent* of Fig. 3 has the usual attributes of the metaclass *SystemComponent*. Moreover, it has the additional attribute *ListOfAudioFiles* which gives the list of audio files. There is also a method *SizeOfAudioFiles()* assigning to each file its size. Other producer classes can be explained similarly. The files produced via objects from the above classes are transferred through interfaces to consumer objects *AudioConsumer*, *VideoConsumer* and *AudioVideoConsumer* which model windows and speakers that broadcast Audio, Video and AudioVideo files into the user environment. They are all created from template metaclasses *SystemComponent*. For example, the *AudioConsumer* class, has the usual attributes of the metaclass *SystemComponent* and models a speaker. As a result, attributes *Volume* and *Tone* of a typical speaker are included.

The next stage which creates *bindobj*s results in the creation of *StreamBindings*, *SignalBindings* and *QoSBindings*. *StreamBindings* deal with transferring continuous media. To check if the QoS of a binding is satisfied the class *QoSBindings* is created which interacts with a control mechanism. Signals are carried through *SignalBindings*.

The user can interact with an IMK through Keyboards and Pointers like mice, trackballs or trackpads. Such devices are modelled via the class *UserInputDevice* which is also created from the metaclass *SystemComponent* of Fig. 3.

The subclasses of the class *Infs* are identical in the CMD and the CCD. Similarly, the classes *ReactiveObject* and *RsigInfs* are identical to the corresponding metaclasses in CMD. After creation of the above classes we need to produce the CMD associations and connect the classes of the CCD together suitably. By creating the association of CMD with the ending *Scs* and *Infs*, the relation between *Infs* classes and other classes of CCD created from *SystemComponent* are depicted. For example, in the CCD, by connecting the class *AudioVideoProducer* to *Infs* with the multiplicity of 1 and 1... we mean that each object of the class *AudioVideoProducer* has a number of interfaces.

5.2 Computational Object Diagram of the Museum Information Kiosk

To produce a model of the computational viewpoint of the Museum Information Kiosk, which is an example of an Information Multimedia Kiosk, [2] presents a heuristic similar to the heuristic explained above.

Applying this heuristic to the Museum Information Kiosk results in a rather large and detailed object diagram. For the purpose of explanation, Fig. 5 depicts the part of the COD which corresponds to the Audio/Video subsystem of Fig. 2. Following the heuristic of [2] one can easily complete the object diagram of Fig. 5 to obtain the full COD of the MIK.

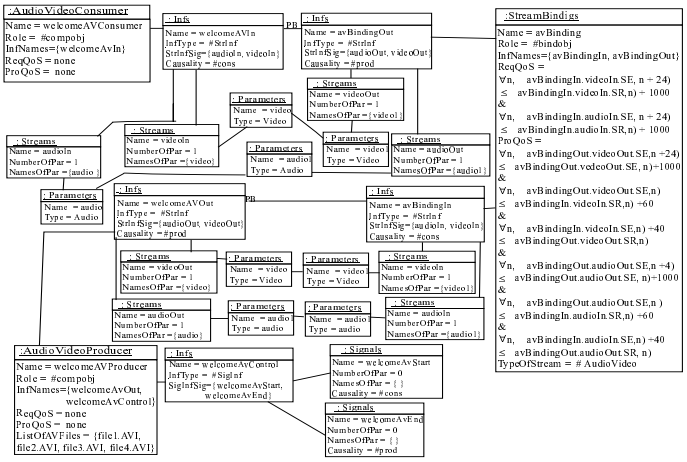

Figure 5: Computational Object Diagram for the video stream

The heuristic [2] for the creation of a COD starts with creation of computational and binding objects. First, objects *welcomeAVConsumer* and *welcomeAVProducer* from the classes *AudioVideoConsumer* and *AudioVideoProducer* are created. Each of these objects includes the attributes of the corresponding class. For example, the object with name *welcomeAVConsumer* is a computational object with *Role* = *#compobj*. It has only one interface *welcomeAvIn* and no provided or required QoS.

Similarly, we need to create binding objects *avBindings* from *StreamBindings*. The provides and required QoS of *avBinding* is as follows. The binding object *avBinding* will provide a throughput of 25 frames per sec and 5 packet per sec, with a delay of between 40 and 60 msec per frames/packets for video/audio, respectively. It must receive the audio frames at a rate of 25

frames per sec and audio packets at a rate of 5 packets per sec from the producer.

The next step is creation of interface objects *welcomeAVIn*, *avBindIn*, etc. from the class *infs*. This follows by instantiation from *Stream* and *Signal* classes. The final stage of the heuristic creates the *Parameter* objects and links corresponding objects together, which in results in the object diagram of Fig. 5.

6 Discussion

There are certain concepts of RM-ODP that cannot be found in UML directly also there are a set of notions in UML that cannot be mapped directly to RM-ODP. A comparison between UML and RM-ODP is drawn in [12]. In the current section we shall discuss some of the differences between UML and RM-ODP as explained in [12] and the way that our approach addresses such issues. We also compare our approach to that of COMET.

Types and classes
The RM-ODP differentiates between types and classes. In the UML, the designer can specify a type as a stereotype of class, which seems a convenient way of modelling. We are currently working towards integrating stereotypes into our research. Stereotypes are particularly helpful for the creation of UML profiles [4], which facilitate use of existing tools. This put a great emphasis on the issue of types. As a result, in this paper, we have implemented the idea of type system of RM-ODP in our UML models. Each attribute and method of our classes or objects or metaclasses must have a type which is either a basic type such as *Integer*, *String*, ... or other types which are defined in RM-ODP such as *video, audio* which are types for streams. This results in models which are called *strongly typed* in the terminology of UML [15]. We use OCL to impose constraints on models [2]. Having a strongly typed model is required for writing OCL expressions as OCL is a typed language. Now, the issue of type checking can be resolved by applying the type checking algorithms of RM-ODP to all attributes and methods of the objects in the model.

Interfaces
An interface in the UML can not be directly instantiated. In our approach, we have modelled interfaces as separate classes *Infs*. As a result, using our heuristics we can create interfaces from such classes. There is an attribute *Name* in *Infs* that, in line with RM-ODP, is an identifier for an interface object. An attribute *InfType* is included that can be used to create different types of interface stream, signal and operational. In addition, all details of different types of interfaces are modelled via subclasses of *Infs* of CMD of Fig. 3 according to the instruction in RM-ODP.

Binding
In the UML there is no direct equivalent for the notion of a binding. The RM-ODP introduces two types of binding: *compound* and *primitive*. The compound binding action of the RM-ODP is performed by a binding object. Compound bindings are modelled as objects which are instantiated from *SystemComponent* in CMD with the role of *bindobj*. Our detailed specification of bindobjs allows us to specify complex real-time systems precisely. A primitive binding action allows binding of two interfaces of the same or different computational objects. The primitive binding between interfaces (*Infs*) is denoted by the self association called *PB* the metaclass *Infs*. In the CMD model of Fig. 3, the primitive binding actions instantiate the association from class *Infs* into itself. The RM-ODP requires the primitive binding to be between interfaces of the same type with complementary causality, which is implemented via an OCL invariant, see [2].

The attribute *Name*
One of the features of our approach, see CMD of Fig. 3, is that we have included an attribute

Name : String in *SystemComponent*, *ReactiveObject*, *Signal*,... referring to the name of such entities. This might seems slightly strange in the eye of a UML modeller, since normally names of objects appear in the first upper box of an instantiated object. The motivation for this is the fact that in the RM-ODP *names are identifiers of corresponding entities*. To emphasise this we have included *Name* as an attributes. This also facilitates writing OCL expressions on the classes and metaclasses.

Layered structure

A central issue in our method is the use of a layered structure CMD, CCD, and COD which is inspired by UML and RM-ODP. We have tried to mimic the way that a modeller will use RM-ODP to produce ODP compliant design. Similarly, the CMD lays the pattern for creation of CODs. Now, it poses a question that, starting from scratch without application of our layered approach, what would the final model of the static aspects of the system look like? To answer this question, we have modelled the MIK via *C*oncurrent *O*bject *M*odelling and architectural design me*T*hod (COMET) [6].

Modelling MIK via COMET

COMET [6] is an object oriented software development process and the full COMET life cycle includes *requirements modelling, analysis, design, construction, integration* and *testing*. The COMET depicts a static model of the system via class diagrams. One of the challenges of the creation of the static model is to identify the concepts involved. To assist the designer in this respect, COMET presents a method called *Object Structuring Criteria* [6], which assists the designer in structuring the system into suitable categories of objects. Fig. 6 represents different category of *application* objects and the hierarchical relationship between them in terms of generalisation between concepts. As a result, each *application* object is either an *interface*, *entity*, *control* or *application logic*. Similarly, each *interface* is either a *user interface*, *device interface* or *system interface*.

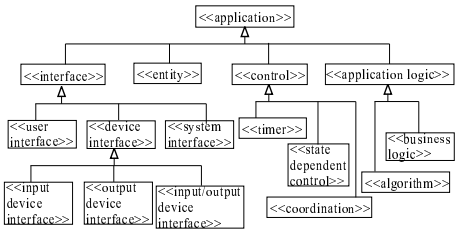

Figure 6: Classification of application classes using stereotypes

Fig. 7 presents a static model of the Museum Information Kiosk (MIK) derived from COMET. The *keyboard* is modelled as an *input device interface*, which means that it is an interface software entity interacting with the hardware device keyboard. *Image Producer* and *Audio/Video Producer* are *output device interfaces* which interact with hardware devices such as monitors and speakers. COMET refers to *voice files*, *images*, *access maps* and *menu* as *entities*. *Local Control* is a *coordinator*, which facilitates lip synchronisation between *Image Producer* and *Audio Producer*. To ensure the correct functionality of the system, a *state dependent control* called *Central Control* is provided. Both *Central Control* and *Local Control* need to be aware of the time of the system, as a result a *timer* class *clock* is included.

Comparison

There are some similarities between our approach and the COMET. The COMET uses diagrams such as Object Structuring Criteria (OSC) of Fig. 6 as a template for identifying the concepts involved and structuring them into class diagrams such as Fig. 7. Similarly, we use the CMD of

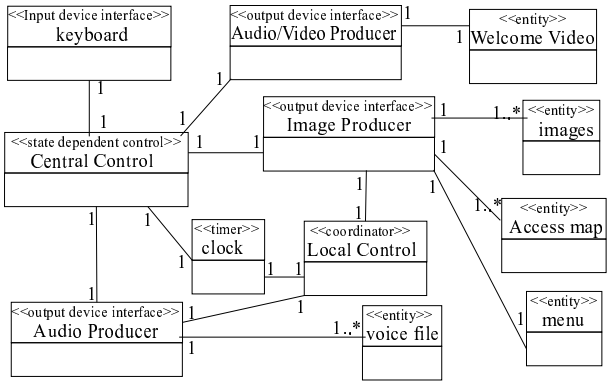

Figure 7: A class diagram for MIK using classification method of COMET

Fig. 3 to produce the CCD of Fig. 4. There are clearly some overlaps between CMD and OSC. However, it seems that the categorisation of concepts in OSC is inspired by engineering aspects of the system. For example, *control*, *application logic*, etc, resemble the components which are expected to appear in the final low level design of the system. We believe that this prevents a COMET user from modelling at a higher level of abstraction. As a result, our approach has the advantage of avoiding the complexity of design in early stages by abstraction into more general categories of objects. Later, relying on heuristics, each high level category can mapped into the CCD classes, which are fine grained objects related to a specific group of applications. The process of such mapping is carried over single classes, which enables the user to avoid dealing with large complex systems.

Being ODP based, our approach rapidly results in a more precise design. For example, COMET does not address the issue of binding as a part of OSC and starting from categorisation of Fig. 6 the designer fails to include bindings. As a result, if the model requires to address issues related to binding, such as QoS for delivery of streams between two interfaces, suitable modelling course of action must be taken in further iteration development stages.

7 Related Work and Conclusion

A number of researchers are working on integrations of UML and RM-ODP. Many of them are interested in UML in the context of enterprise specification. Aagadel and Milosevic [1] study the way that enterprise viewpoints can be used as a part of software development cycle via UML. They also elaborate on different enterprise modelling concepts in terms of UML. Linington [11] uses UML to specify the enterprise viewpoint specifications of RM-ODP. Steen and Derrick [13] present a metamodel UML core for the enterprise viewpoint in the RM-ODP and also study the extent that UML can be applied for the specification of the enterprise viewpoint.

Other papers concentrate on specific applications. Oldewick and Berre [14] apply a methodology based on RM-ODP and UML to geographical information systems. Kande et al. [9] apply UML to specify service components of telecommunications management networking systems which are originally modelled in the framework of RM-ODP.

In this paper, we have applied the method of [2] to an Interactive Multimedia Kiosk example. We have presented a class diagram from which UML models of different IMK applications such as MIKs can be created. Our case study demonstrates that our approach is successfully applicable

to systems containing multiple types of media. Our example in [2] illustrated simple audio and video channels; the case study of this paper encompases a larger range of media types such as text, graphics, animation and sound files. We have also addressed the issue of extensibility by demonstrating that the method is applicable to large examples. It is straightforward to follow the method explained in the paper to present a UML model of a central database system interacting with more than one IMK.

Doing the case study, we observed that the challenging part of the implementation of our process is the creation of the Computational Class Diagram (CCD). Since creation of the CCD requires a detailed knowledge of the Specific Domain of Application (SDoA). For example, to instantiate the CCD of the IMK, we have to know what *compobj*, *bindobj*, ... are involved and what their attributes are. In other words, like any other modelling task, it is essential to obtain a deep understanding of the SDoA. However, our method has the advantage of directing the modeller to search for the information and also recording such information in UML diagrams in a manner which complies the RM-ODP. The heuristic nature of our approach suggests that there is clear scope to develop tailored tools based on existing UML tools.

The paper has made a comparison between UML and ODP and shows that the approach [2] bridges the gap between specification via UML and ODP. The paper also applies a non-ODP (COMET) approach to the modelling of the application and draws a comparison with our method. We argue that our approach not only results in a more precise models, but also it avoids the complexity of design by starting at a higher level of conceptual modeling and follows the refinement of the model focusing on smaller and more manageable components of the system.

We are currently working on creation of a UML profile [4] which implements our method. In future we are planning to address the behavioural aspects of such distributed systems and also apply our approach to modelling of multicasting.

References

[1] J. O. Aagedal and Z. Milosevic, *ODP Enterprise Language: UML Perspective*, 3rd International Enterprise Distributed Object Computing Conference (EDOC'99), Mannheim, Germany, pp. 27-30, 1999.

[2] B. Bordbar, J. Derrick and A.G. Waters, *Using UML to specify QoS Constraints in ODP*, accepted for publication in Computer Networks and ISDN Systems.

[3] G. Blair and J.- B. Stefani, *Open Distributed Processing and Multimedia*, Addison-Wesley, 1997

[4] S. Cook, *The UML Family: Profiles, Prefaces and Packages* Proceeding of UML 2000-The Unified Modelling Language: Advancing the Standard (October 2000), pp. 255-265.

[5] M. Cranston, D. J. Clayton and P. J. Farrands *Design and Implementation Consideration for an Interactive Multimedia Kiosk: Where to Start*, Proc. of the 13th Annual Conference of the Australian Society for Computer Learning in Tertiary (ASCILTE), Adelaide S. A., pp. 101-112.

[6] H. Gomaa, *Designing Concurrent, Distributed and Real-Time Applications with UML*, Addison-Wesley Object Technology series, 2000.

[7] *ITU Recommendation X.901-904 ISO/IEC 10746 1-4*. Open Distributed Processing Reference Model - Parts 1-4, July 1995.

[8] F. Jahanian and A. K.-L. Mok, *Safety Analysis of Timing Properties in Real-Time Systems*, IEEE Trans. Software. Eng., vol. 9 No. 12, pp. 890-904, 1986.

[9] M.M. Kande, S. Mazaher, O. Prnjat, L. Sacks and M. Wittig, *Applying UML to Design an Inter-Domain Service Management Application*, UML 98, LNCS 1618, pp. 200-214, Springer, 1999.

[10] P. F. Linington, *RM-ODP: The Architecture*, In K. Raymond and E. Armstrong, editors, IFIP TC6 International conference on Open Distributed Processing, pp. 15-33, Brisbane, Australia, February1995. Chapman and Hall.

[11] P. F. Linington, *Options for expressing ODP Enterprise Communities and their Policies by using UML*, Proceedings of the Third International Enterprise Distributed Object Computing Conference, pages 72-82. IEEE, September 1999.

[12] J. R. Putman, *Architecting with RM-ODP*, Prentice Hall, 2001.

[13] M.W.A. Steen and J. Derrick, *ODP Enterprise viewpoint specification*, Computer Standards and Interfaces, Vol. 22, 165-189, 2000.

[14] J. Oldevik and A.-J. Berre, *UML based methodology for distributed systems*, 2nd International Enterprise Distributed Object Computing Conference (EDOC 98) San Diego

[15] *UML 1.3 Documentation*, Rational Rose Resource Centre, 1999.