

Building a UML Profile for MultiTEL

Lidia Fuentes, José M. Troya, and Antonio Vallecillo
ETSI Informática. Universidad de Málaga
{lff,troya,av}@lcc.uma.es

Abstract A UML profile for a system or an application is a standard means for expressing the semantics of this system or application using a set of predefined extensions to UML. UML profiles allow all stakeholders of a system to share a common graphical notation and vocabulary, and permit more precise specifications and better documentation on how to use and customize systems. In this paper we present a UML profile for MultiTEL, a framework particularly well suited for the development of multimedia and collaborative applications in open distributed environments, such as the Internet. We show how to build such a UML profile, and how systems designers can use it to derive and document application frameworks for multimedia applications.

1. Introduction

The World Wide Web and private Intranets are increasingly perceived as the natural way to access Internet services –specially multimedia and cooperative services. These kinds of applications require distributed access and processing, which is not directly supported by Web technologies yet. The current approach to deal with this problem is the use of distributed component platforms (DCPs) and middleware platforms, that support the coordination of distributed components.

Although current DCPs, such as CORBA, J2EE, or .NET, have become an interesting alternative for integrating heterogeneous distributed components in short time, they also present some important limitations when tried to be used for building and reusing component-based applications.

- (a) DCPs provide coordination mechanisms for individual components to interoperate, but they do not care about the global configuration of the application, i.e., its software architecture.
- (b) DCPs provide methods and mechanisms for the deployment, reuse, and communication of heterogeneous components, but based on their syntactic interfaces only. Semantic information about the components and the application is missing, or written in textual documents for human readers only.
- (c) DCPs provide rudimentary environments for component assembly, but do not provide high-level formal reasoning for the definition, specification, documentation, and customization of reference architectures.
- (d) Finally, DCPs do not provide any methodologies for the derivation of components and distributed applications in their respective environments.

Application Frameworks (AFs) define a technology that try to provide answers to some of these problems, and have become one of the cornerstone technologies for the effective development of reusable systems and applications. AFs have been defined as a reusable design of all or part of a system being represented, and have demonstrated the usefulness of encapsulating a reusable, customizable software architecture as a collection of collaborating, extensible components [1]. Usually built upon DCPs, such AFs are particularly important for developing open systems, as the Web is, in which not only functionality but architecture can be reused across a family of related applications [2]. Unfortunately, they also present some limitations, specially when it comes to documenting and reusing them. This problem, known as the *documentation* problem [3], refers to the difficulty of documenting an AF in order for developers to decide whether it suits their necessities or not, and if so, how to customize it to fulfill their particular requirements. One of the root causes of this problem is the insufficient information available about the architecture of the AF, which really difficulties its understanding.

The lack of effective application documentation has not been ignored by the software engineering community. What we have seen in the last few years, especially as a result of efforts at OMG and W3C, is a gradual move to more complete semantic models as well as data representation interchange standards, that try to address this documentation problem. In particular, the Unified Modeling Language (UML) has been widely adopted as the common language for designing and modeling all sort of systems and applications. In addition to UML, OMG contributions include XMI (XML Metadata Interchange), MOF (Meta-Object Facility) and CWM (Common Warehouse Metamodel). W3C contributions include XML, XML Schema, and the ongoing work of XML-PC working group. These technologies can be used to more completely integrate the value chain (or life cycle) when developing and deploying component based applications for various target software architectures.

OMG's current plans try to change the fact that historically the integration between the development tools and the deployment into the middleware frameworks has been weak. This is now beginning to change by using key elements of the OMG's new Model Driven Architecture (MDA), namely specific models and XML DTDs that try to cover the complete life cycle of software systems, as well as *profiles* that provide mappings between the models used in various life cycle phases. The facilities provided by XMI for data exchange between different models are used in the MDA for marrying the worlds of modeling (UML), metadata (MOF and XML) and middleware (UML profiles for CORBA, Java, EJB, etc.). XMI also provides developers focused on implementation in Java, VB, HTML, etc., a natural way of taking advantage of the software architecture and engineering discipline when a more rigorous development process is desirable [4]. One of the key elements in this chain is the use of UML for modeling systems.

UML models are declarative models, as are IDL-based object models, Java interfaces, and Microsoft COM and .NET IDL interfaces. However, UML models can be expressed visually, and they are semantically much richer. Their visual representation allows the stakeholders of a system (e.g., owners, users, developers, administrators, and maintainers) to share a common graphical notation that can be understood by all parties. UML facilities for incorporating richer semantic information into the system model allows more precise specifications, avoids many misunderstandings and ambiguities, and permits better documentation, which enhances the understandability of how the system works, how to use it, and how to reuse it, i.e., how to customize the system, its structure, or parts of it.

Based on UML, and trying to make use of all the benefits and tools that UML provides, OMG has also defined UML profiles. A *UML profile* is a set of extensions to UML using the built-in extension facilities of UML. A UML profile for a DCP or an application framework is a standard means for expressing the semantics of that DCP or AF using UML. A number of UML profiles are currently at various stages of standardization by OMG (UML profile for CORBA is already adopted [5]). They try to provide the critical links that bridge the UML community (model based design and analysis) to the developer community (Java, VB, C++ developers), the middleware community (CORBA, EJB, SOAP developers), etc.

In addition to the specification of UML profiles for DCPs, UML profiles focused on systems and applications are also needed [4]. With them, applications could be designed and documented so all the previously mentioned benefits could be obtained for them, too. In particular, application frameworks are the natural candidate for benefiting from the potential advantages that the use of UML profiles could provide, thus helping to alleviate the documentation problem mentioned above.

In this paper we present a UML profile for a particular application framework, called MultiTEL [6,7], specially well suited for the development of multimedia and collaborative applications in open distributed environments, such as the Internet. We will show how a UML profile can be built for MultiTEL, and how systems designers can use this profile to derive and document application frameworks for multimedia and collaborative applications.

The structure of this paper is as follows. After this introduction, section 2 describes the notion of UML profiles and their potential benefits. Section 3 briefly introduces MultiTEL, describing both its compositional model and the framework it defines for building multimedia applications. The UML profile for MultiTEL is presented in section 4, and then used in section 5 to discuss how it can be used for building UML models for multimedia application frameworks. An example is used to illustrate this approach: a distributed VoD service. Finally, section 6 draws some conclusions and outlines some further research activities.

2. UML profiles

In general, a *UML profile* is a set of extensions to UML using the built-in extension facilities provided by UML (stereotypes, tagged values, and constraints). There is no OMG normative definition of a UML profile yet, although the OMG's Business Object Initiative elucidated the following working definition of a UML profile:

A *UML profile* is a specification that does one or more of the following:

- ?? Identifies a subset of the UML metamodel (which may be the entire UML metamodel).
- ?? Specifies "well-formedness rules" beyond those specified by the identified subset of the UML metamodel. "Well-formedness rule" is a term used in the normative UML metamodel specification [8] to

describe a set of constraints written in natural language and UML's Object Constraint Language (OCL) that contributes to the definition of a metamodel element.

- ?? Specifies "standard elements" beyond those specified by the identified subset of the UML metamodel. "Standard element" is a term used in the UML metamodel specification [8] to describe a standard instance of a UML stereotype, tagged value or constraint.
- ?? Specifies semantics, expressed in natural language, beyond those specified by the identified subset of the UML metamodel.
- ?? Specifies common model elements (i.e., instances of UML constructs), expressed in terms of the profile.

For instance, the UML Profile for CORBA specification [5] has been devised to provide a standard means for expressing the semantics of CORBA IDL using UML notation, and thus to support expressing these semantics with UML tools. A model defined in the UML profile for CORBA is an alternative representation of an IDL model –from which the original IDL model can be derived– that provides additional semantics (such as cardinality, or pre/post conditions on operations) that cannot be represented in an IDL model today. Thus, with the UML Profile for CORBA, CORBA-based specifications can be made much more rigorous than is possible with IDLs only. On the other hand, appropriate extensions to the CORBA IDL, that allow representation of these additional relevant concepts, would make it possible to map a model expressed in the CORBA profile of UML to an equivalent IDL model in a reversible fashion. That is, one would be able to reconstruct the corresponding UML from the equivalent IDL, without loss of information. This ability to "round-trip" the transformation in this way would allow designers and architects to work in the technology that they are comfortable with (UML or IDL) and algorithmically generate the alternative representation for the specification [4].

Another advantage of UML models is that they can model not only platform specific applications, but they can also be *platform-independent*, i.e., UML can be used to state fundamental semantics without committing to a specific platform. Thus, a *platform-independent model* is a formal specification of the structure and functionality of a system that abstracts away technical details. Such platform-independent models are useful for two basic reasons. Firstly, by distilling the fundamental semantics they make it easier to validate the correctness of those semantics. Platform-specific semantics cloud the picture in this respect. Secondly, they make it easier to produce implementations on different platforms while holding the essential semantics of the system invariant, by making an unambiguous description of the semantics available. The great benefit is *portability of specifications* from one language environment to another, as well as *portability of implementations* among different instances of the same language environment [4].

Building a UML profile for MultiTEL would provide us with the same sort of benefits. First, it would provide an enhanced documentation for MultiTEL mechanisms and concepts, allowing users to get a better understanding on how it works, how to use it, and how to customize it. Second, it would allow designers and architects of MultiTEL frameworks and applications to use UML for designing their systems, being able to transform their UML designs into correct MultiTEL systems. And finally, it would allow the representation of existing MultiTEL systems in UML, providing highly readable and semantically rich documentation for those systems, independently from how and where they are implemented.

3. MultiTEL

MultiTEL (Multimedia TELecommunication services) is a complete development solution that applies component and framework technologies to deploy multimedia and collaborative applications [6,7]. Essentially, MultiTEL defines both a *compositional model* implemented as a composition platform, and an *application framework* for developing multimedia applications for the Web.

3.1 The MultiTEL component model

Describing software architectures in terms of the relationships between components and their interactions brings us closer to a compositional view of systems engineering. Although everybody agrees that component-oriented programming represents a significant advance towards the development of systems by assembling already tested and validated off-the-shelf components (COTS) –thus reducing development costs and efforts, and simplifying designs–, there is a lack of programming languages that support components and composition rules as language primitives.

On the other hand, distributed component platforms provide a component abstraction and a (more or less dynamic) composition mechanism that allow the easy integration of heterogeneous distributed components. However, as mentioned earlier, the application architecture definition is not explicitly stated anywhere, but usually spread throughout the component implementation modules. Consequently, high level composition models are needed in order to endow component platforms with architectural abstractions, as well as object-oriented languages with assembly capabilities beyond inheritance [9]. This approach seems very promising in order to improve the quality, flexibility and adaptability of applications, helping to develop complex and distributed applications in shorter time.

It is commonly agreed that the separation of concerns principle is a good approach to support adaptability and architectural evolution. This principle enforces the separation of different aspects of a system, for instance the coordination and computation issues. In this case, a change in a coordination protocol can be simply achieved by replacing the control component, without modifying the computation involved.

The MultiTEL compositional model provides this separation of concerns between computation and coordination by encapsulating them into two different kind of entities. The compositional units of the model are *Components*, that encapsulate computation, and *Connectors*, that encapsulate the coordination protocols. In MultiTEL, coordination protocols are specified by state transition systems, which are expressed by state transition diagrams. The set of connections between the components and connectors of an application defines the architecture of the application. Those connections are governed by a set of plug-compatibility rules in order to guarantee the correct interoperation of components, and to avoid interaction problems.

The connection between components and connectors is dynamically established in MultiTEL at run time, providing a powerful mechanism for late binding among them. This is very useful for implementing applications in open systems, in which components evolve over time, or applications with dynamic topologies, such as Web applications. Thus, MultiTEL defines a dynamic binding mechanism that plugs components into connectors at runtime, providing the *glue* between the computation and coordination aspects of the application in a very flexible way. This dynamic composition mechanism is modelled by the *LocalUSP*, an entity that maintains the definition of the architecture of the application, in terms of the constituent components, the connectors, and the connections among them. The LocalUSP plays the role of a component platform used by components and connectors to resolve their interconnections are run-time, and constitutes the base technology for any application frame work built with MultiTEL.

3.2 The MultiTEL framework

In general, the MultiTEL compositional model can be used for developing any distributed application that wants to make use of the separation of concerns provided by MultiTEL, by means of component and connectors. But not only applications can be built on top of that compositional model, but also it can be used to design and develop application frameworks. In particular, MultiTEL has been successfully used for building an application framework for the derivation of all sort of multimedia applications, such as VoD, tele-education, chats, video-conferences, etc [6].

This approach implies two steps. First, the description of a *generic software architecture*, in terms of abstract components and connectors. And second, the implementation of some of those abstract components and connectors, which will be the ones that constitute the framework. In this sense, multimedia application builders could have access to a set of reusable software components, which greatly facilitate the services implementation, while hiding from the service designer the details of underlying technologies and the complexity of interaction patterns. For instance, the IDL interface of a *Camera* component does not depend on the hardware technology used by the actual camera, which may be implemented by both a Webcam or a network camera. Likewise, the connector that controls the data read/write operations for the reproduction of a movie always implements the same protocol, independently from the multimedia data format used (JPEG, MPEG, ...).

The MultiTEL framework defines a set of interrelated components and connectors, where components model real entities in multimedia services (participants, multimedia devices, authentication databases, ...), and connectors characterize the dynamic relationships among those components. Framework components and connectors reside inside a repository, that can be extended by adding new component and connector IDLs (e.g., new devices), or by adding new implementations of those components and connectors (e.g., new models of cameras).

4. A UML profile for MultiTEL

The aim of designing a UML profile for MultiTEL is to provide a standard means for expressing the semantics of MultiTEL entities using UML notation and thus to support expressing these semantics with UML tools. Furthermore, using the UML profile for MultiTEL allows the design of systems, applications, and even new application frameworks (e.g., for the derivation of collaborative applications) independently from the underlying component platform (Java/RMI, EJB, CORBA, etc.). For instance, the current version of the MultiTEL framework is implemented using Java/RMI¹, although by using the UML profile for MultiTEL we will be able to abstract away these implementation details, and concentrate just in the architecture of the system and its semantics.

Of course, any multimedia service derived from MultiTEL could be modelled with plain UML object abstractions. However, using an standard UML extension that precisely captures the MultiTEL component model would provide standard documentation, standard design mechanisms, and at an appropriate level of abstraction. This would greatly benefit the portability and reuse of the system specifications and designs, and produce richer and more useful documentation of the system, which helps alleviating the documentation problem mentioned in the introduction.

4.1 The UML profile

Following the process for defining a general UML profile (described in section 2), and the special characteristics of MultiTEL, we have distinguished two main levels of abstraction in the definition of the UML profile for MultiTEL. At the higher level we have the *Virtual Metamodel*, that defines the basic entities of the model, namely the component, the connector, and the LocalUSP, in terms of the corresponding UML stereotypes. These entities will be the ones used in the second level, the *model*, that defines the class diagrams that specify the compositional model of MultiTEL, i.e., the structure and relationships between the model entities. Finally, we will show these two levels can be further refined into a third one, that corresponds to the UML profile for an application framework built with MultiTEL. For this we will use an example, an application framework for the implementation of VoD services in an open environment.

4.2 The UML MultiTEL Virtual Metamodel

The MultiTEL virtual metamodel resides in the top layer because it represents the model entities at the highest level of abstraction. A virtual model describes data belonging to the layer below the metamodel. Thus, the first level contains the definitions of the UML stereotypes for MultiTEL primitive entities, namely components, connectors, and the LocalUSP. These stereotypes will be used in the second level where the MultiTEL model is specified.

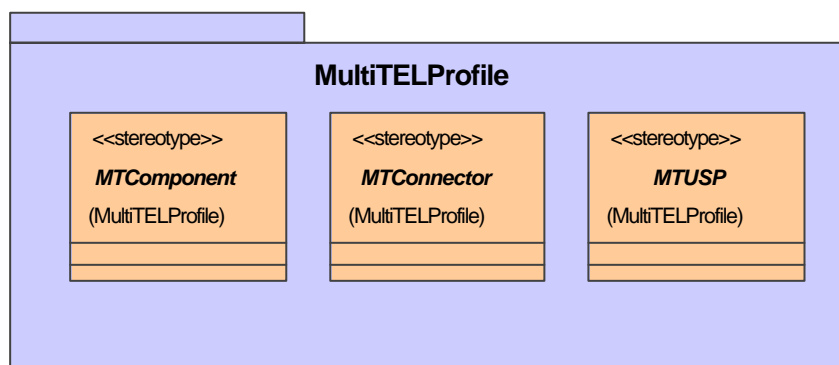


Figure 1. Virtual metamodel for MultiTEL.

Since MultiTEL is an architectural model defined independently from any component platform or programming language, the virtual metamodel does not need to define any predefined types, nor the stereotypes need to have any dependency relationship with any UML metaclass. Thus, we only need to define at this level

¹ Available at <http://www.lcc.uma.es/~lff/MultiTEL>

the names of the MultiTEL stereotypes that are going to be used in the MultiTEL model definition. This proposal takes the following approach to using UML notation to express this virtual metamodel:

- ?? The model will be expressed via class diagrams.
- ?? Each stereotype does not has a dependency relationship with any UML metaclass.
- ?? Each stereotype is expressed by a class box, even though a stereotype is not a user defined class. The keyword <<stereotype>> does not represent a sterotype itself. It is simply a notational marker for the underlying stereotype metaclass.

With all this, the virtual metamodel for MultiTEL is the one depicted in Figure 1.

4.3 The UML model for MultiTEL

In this section we are going to describe the class diagrams that make use of the stereotypes defined in the previous section. All properties of UML metamodel elements contained in the UML profile may be used to specify an object model that conforms to the profile. So, we are going to use the facilities of UML for describing attributes, operations and aggregation properties of associations. The intention is to define the class diagrams that specify the compositional model of MultiTEL by using the entities in the metamodel, specifying the structure and relationships between the model entities.

Figure 2 shows the class diagram of the MultiTEL model. The main entities of the model are of course the component, the connector and the LocalUSP, but at this level they are defined with the corresponding stereotypes keywords, *MTComponent*, *MTConnector* and *MTUSP*.

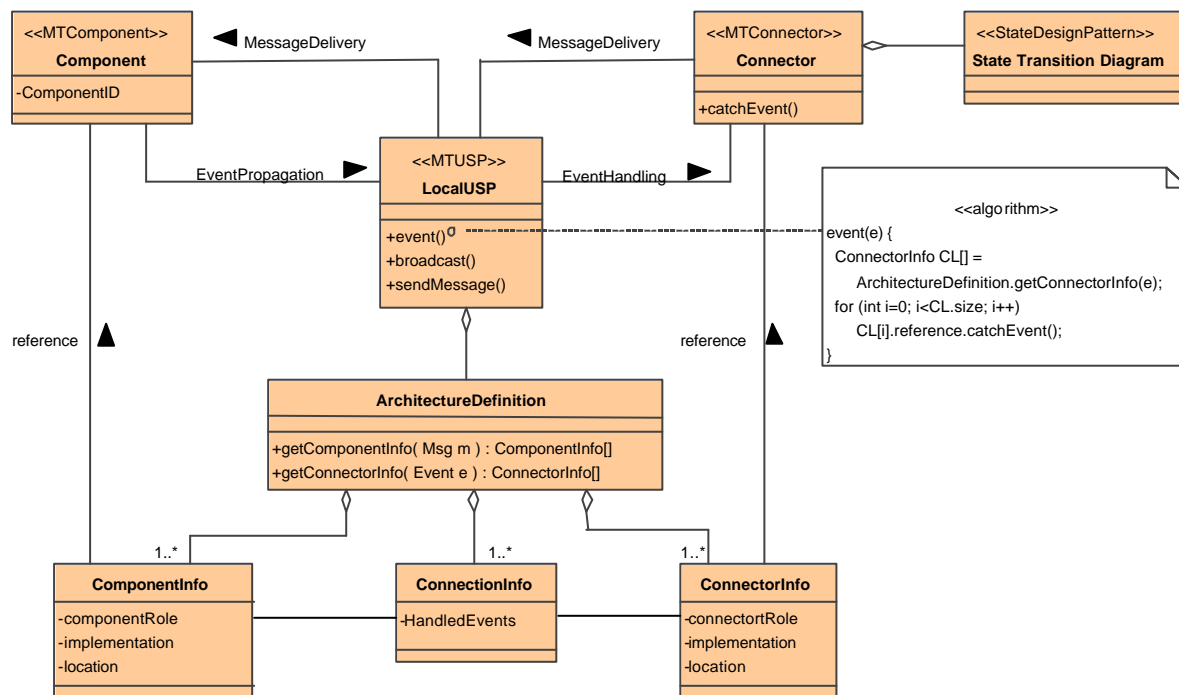


Figure 2. The UML model for MultiTEL

Components have an attribute, *ComponentID*, that designates a unique global identifier, which serves as a unique address for component instances. This identifier is the one used by connectors for identifying the concrete component instances to which send responses to. Components are independent computational entities, and therefore they do not include any information or reference about other entities of the system. A component is able to receive messages and react to them by performing some internal computation. Those internal computations usually terminate by “throwing” an event to the system, using the *event()* primitive defined in the *LocalUSP* class. Please notice how components throw events to the *LocalUSP* instead of naming particular connectors, because components are completely unaware of the different interactions in which they are engaged in at any moment in time. In MultiTEL, components handle messages and connectors handle events. The *LocalUSP* class is the responsible of finding the appropriate connector or connectors that should handle an event,

as explained below. Likewise, the *LocalUSP* is responsible for the actual invocation of a component operation, originally requested by a connector. In the diagram in Figure 2, reception of messages is modeled by the *MessageDelivery* association, and propagation of events by the *EventPropagation* association.

Connectors encapsulate synchronization protocols, which are defined by state transition diagrams (STD). The *Connector* class handles the events by means of the `catchEvent()` operation (shown in Figure 2 by the *EventHandling* association), which is invoked by the *LocalUSP* and that triggers its state transition diagram. We recommend that the SDT implementation follows the *State* design pattern [10], and so the *State Transition Diagram* class has been labeled with the stereotype keyword *StateDesignPattern* as a way to express this suggestion. A connector can be (indirectly) related to one or more components through the *EventHandling* association, because it catches and handles the events propagated by them. This relation is done through the *LocalUSP*, which maps the event propagations from components to the event handling by the appropriate connectors. Connectors react to these events by sending messages to components, which is described by a *MessageDelivery* association in Figure 2. Again, the *LocalUSP* class is in charge of the late binding of these associations, since it is the entity that contains the information about the architecture of the application, i.e., how components have to be plugged into connectors.

We can consider the *LocalUSP* as a global configuration class that performs the runtime composition of components and connectors according to the application architecture. Frameworks that encapsulate a reusable, tailorable software architecture are particularly important for developing open systems in which not only functionality but architecture is reused across a family of related applications [2]. One significant contribution of MultiTEL is that it defines architectural information inside the *LocalUSP* class, thus making application architecture explicit.

An application architecture (AA) in terms of MultiTEL consists of a set of components, a set of connectors and a set of static or dynamic links that may be established during the application execution. In the UML profile, the AA is defined as the *ArchitectureDefinition* class, which is part of the *LocalUSP*, and that stores the relevant information about the components and the connectors that form part of the application, and about the connections among them. This information is represented in the UML model by classes *ComponentInfo*, *ConnectorInfo* and *ConnectionInfo*, respectively.

The first two classes describe components and connectors, as indicated by the *describes* associations in Figure 2. For each component and connector in the architecture, the following information is maintained:

- ?? *Implementation*. Useful information for creating and deploying a component instance, such as component and connector implementation class names, installation requirements and instructions, etc.
- ?? *Location*. Information concerning the distribution of components and connectors, such as the physical location where they reside.
- ?? *Role*. This is the name of the component or connector from the architectural point of view. Each component/connector is registered with a name within a particular architecture, that serves to identify the “rôle” its plays within the application. Examples of roles are “participant”, “manager”, “resource”, etc. The possible names for roles depend on the application domain. A given component or connector may play different roles in different applications.
- ?? *reference*. This association takes place when the component or connector has already been instantiated.

Class *ConnectionInfo* encapsulates the information relevant to each connection in the system, in terms of a component role, a connector role, and the list of events that a given connector performing the connector role should be able to handle from a component performing the component role. When building and deploying a particular application, its components, connectors and the connections among them should be supplied by assigning values to objects of these classes.

Finally, operations `getComponentInfo()` and `getConnectorInfo()` are used within the *LocalUSP* to interrogate class *ArchitectureDefinition* about the components and connectors associated to a message or an event, respectively, in order to carry out the actual deliveries. We have said that the *LocalUSP* class encapsulated the dynamic binding information between components and connectors. Its `event()` operation is accessed by components for propagating an event. When the *LocalUSP* receives an `event()` operation, it looks up in the *ArchitectureDefinition* the set of connectors supposed to handle the propagated event for this application. And then, the *LocalUSP* invokes the `catchEvent()` operation on the target connectors. This dynamic binding mechanism may be considered as a kind of ORB service of the middleware platform provided by MultiTEL, and the algorithm that defines its behavior has been described in the class diagram as a note with the `<<algorithm>>`

stereotype. Please note the benefits of the UML profile, since it allows to specify not only syntactic information and class structures, but also some semantic information about the behavior of the system being modeled. Likewise for message delivery, connectors may invoke `sendMessage()` or `broadcast()` operations of *LocalUSP* to send messages to components (i.e., invoke their operations). Due to the distributed nature of MultiTEL and the nature of the Web applications it models, a mechanism for broadcasting messages to all suitable components has been defined, in addition to the traditional one-to-one client-server invocations. Again, the communication between the connectors and the components is not direct in order to properly implement the late binding facilities. In Figure 2, the algorithms that rule the behavior of those two last methods have been omitted.

5. An example application

So far we have defined the virtual metamodel and the representation of the MultiTEL compositional model, which is the base of all MultiTEL applications. However, one of the major advantages of MultiTEL is its ability to design and implement application frameworks for particular families of systems, such as multimedia and collaborative applications. In this section we will show how the UML profile for the MultiTEL model can be *refined* to design an example application framework, in particular one that provides a distributed video on demand (VoD) service. For the sake of simplicity we will keep the example as small as possible, just considering the reproduction of a movie file of any given format into a display window.

In order to design the system, we need to identify first the components and connectors of the application, and then specify its structure (shown in Figure 3). The classes defining the application entities will inherit (in the UML sense) from the *MTCComponent* and *MTCConnector* classes in the MultiTEL model, respectively.

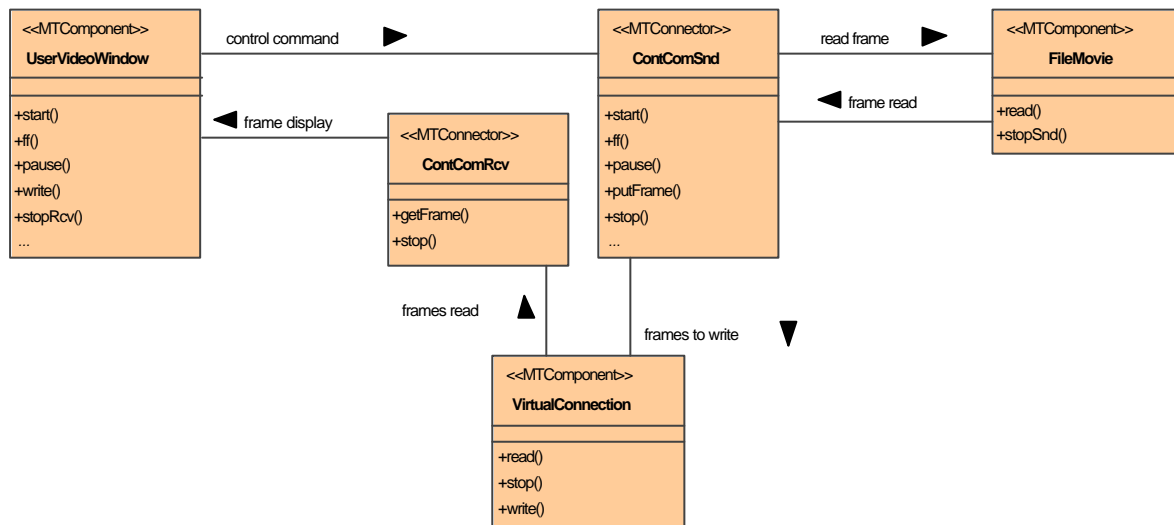


Figure 3. The VoD model specified according to the MultiTEL model.

There are three kinds of components: *UserVideoWindow*, *FileMovie*, and *VirtualConnection*. The first one models the user's Window for displaying the movie and controlling its reproduction (via the `start()`, `ff()`, `pause()`, ... operations). The *FileMovie* component models the video movie, independently from its data format and storage location. This component receives requests for reading frames, and delivers them by means of events. The last component is the *VirtualConnection* component, which is in charge of establishing the connection between the movie file and the user window, independently from the protocol used (TCP, Multicast IP, Real Time Protocol, etc.). Intuitively, it acts as a pipe between the user and the movie for reading and writing multimedia data. There will be one instance in the user site and another one in the machine that stores the movie, both communicated by a kind of socket.

The two connectors control the read/write operations, and have information about the roles names of the source and target components for plugging them in. The *ContComSnd* reacts to the user control events, decides the next frame to read, and passes the resulting frame (obtained from the *FileMovie* component as an event) to the appropriate *VirtualConnection* component. The *ContComRcv* connector is the one in charge of receiving the

video frame at the user's site, and delivering them to the corresponding user window component. Again, frames are delivered to connectors as events, and transmitted to components as messages. All communications are managed by the *LocalUSP*, which contains the actual architecture of the application and acts as a mediator between the components and the connectors.

6. Conclusions

The Web now offers an exceptional infrastructure for the development of open distributed services and applications. However, most of the existing web-based systems only make use of its access and visualization facilities, but without exploiting its distributed access and processing potential capabilities. Furthermore, specific technologies –beyond client-server architectures– for the design and development of complex web-based applications are also lacking.

An interesting possibility to consider is the use of Object-Oriented design techniques and methodologies which are now successfully being used for the development of many industrial applications. In particular, we refer to the use of UML for system modeling, DCPs for composition and coordination of distributed components, and design patterns and application frameworks for designing and developing Web-based services and applications. However, some extensions of those concepts are needed in order to cope with the specific characteristics of Web-based systems, and to overcome some of their current limitations.

In this paper we have explored the use of UML profiles for expressing the semantics of application frameworks. In particular, we have shown how a UML profile can be built for MultiTEL, and how it can be used for designing Web-based multimedia applications. The advantages of the approach have already been outlined, but what we have basically obtained with this UML profile is a standard way to design and document systems and application frameworks, which makes use of the benefits of UML models: visual representation, easy to learn and to use by most system stakeholders, richer semantic descriptions, enhanced, and supported by a number of industrial tools.

There are, however, some open issues that need to be investigated further. In the first place, we think that platform-independent UML profiles are not enough, specially for application frameworks. The advantage of using application frameworks is not only about reusing a software architecture, but also about reusing code. A common definition describes an application frameworks as a “partially instantiated application architecture”. Thus, application frameworks are semi-developed applications, that leverage the user to implement most of the common services, facilities and functionality of the applications. In this sense, platform-independent models are useful but insufficient for documenting application frameworks, since some details about the AF platform-specific implementation needs to be described, too. Of course, those platform-specific models can be designed using the appropriate UML profile for the corresponding component platforms (e.g., CORBA, EJB, ...), providing an appropriate level of abstraction for them. However, the current UML's facilities for relating models at different levels of abstraction are rudimentary and need expansion; in particular, UML's semantics do not adequately support the powerful zoom-in and zoom-out capabilities needed in this case [4]. Realistically, the mapping between platform-independent and platform-specific models cannot be automated yet, it needs to be done by (error-prone) humans. We need to gain more experience in order to define various standard patterns and allow them to be selected in some way, allowing to automate this process.

Another issue is about interoperability. In theory, XMI and the MOF should provide the bridges between models developed using different modeling methods and tools. This would iron out many migration problems in current organizations, that cannot simply build new systems and applications starting from scratch but have to take into account their legacy systems. Likewise, systems are no longer designed in isolation, they usually form part of a bigger system or organization, and they have to interoperate with other external systems (the ones used by the company's suppliers, clients, etc.). We cannot make the simplistic assumption that all systems will be designed and modeled using the same modeling tools. Therefore, bridges between models are needed, although the problem is that the bridges currently provided by XMI and the MOF are immature and insufficient.

The final issue is about the formal semantics of UML, that need to be made precise in order to provide more useful semantic information of the applications. Current semantics of UML are quite loose, although there are several ongoing initiatives for making UML precise, e.g., the ones by the “precise UML” group, <http://www.cs.york.ac.uk/pUML>, and also the ones by the OMG, with a RFP in process called “Action semantics for UML” that tries to move UML beyond first order (declarative) semantics into second order (imperative) semantics. The idea is to produce executable UML models in a more feasible way than it is possible today.

Summarizing, it seems that we are in the right track, but there is still a long way ahead if we want to be able to use all the potential capabilities of the Web for the development of really open distributed services and applications.

References

1. Mohamed E. Fayad and Douglas C. Schmidt. "Object-Oriented Application Frameworks". *Communications of the ACM*, 40(10):32–38, October 1997.
2. Serge Demeyer, Theo D. Meijler, Oscar Nierstrasz, and Patrick Steyaert. "Design Guidelines for 'Tailorable' Frameworks". *Communications of the ACM*, 40(10):60–64, October 1997.
3. S. Sparks, K. Benner, and C. Faris. "Managing Object-Oriented Framework Reuse". *Computer Journal*, 39(9):52–61, September 1996.
4. OMG. "Model Driven Architecture. A Technical Perspective". OMG document ab/2001-01-01, January 2000.
5. OMG. "UML Profile for CORBA Specification. V1.0". Final adopted specification. OMG document ptc/00-10-01, October 2000.
6. Lidia Fuentes and José M. Troya. "MultiTEL: Multimedia Telecommunication Services Framework". In *Domain Specific Application Frameworks: Frameworks Experience in Industry*. Wiley & Sons, October 1999.
7. Lidia Fuentes and José M. Troya. "A Java Framework for Multimedia and Collaborative Applications over a Web-based Platform". *IEEE Internet Computing* 3(2):55–64, March-April 1999.
8. OMG. "UML Metamodel Specification". OMG document ad/99-06-08, June 1999.
9. Clemens Szyperski. *Component Software. Beyond Object-Oriented Programming*. Addison-Wesley Longman, 1998.
10. Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.