MUSIC: an interactive MUltimedia ServIce Composition environment for distributed systems.

Piergiorgio Bosco, Giovanni Martini, Giovanni Reteuna

CSELT - Via G.Reiss Romoli 274, 10148 - Torino - Italy

{Piergiorgio.Bosco,Giovanni.Martini}@cselt.it

Keywords: Distributed Multimedia Applications Development, IMA, CORBA, TINA

Abstract - The objective of MUSIC is to devise a development environment for the composition of distributed multimedia applications over CORBA platforms, that uses a graphical model to support their specification. A movie-like approach has been adopted to assemble both processing service objects and physical networked devices.

1. Introduction

The distributed multimedia systems combine in an integrated way the advantages of the distributed elaboration [13] and the possibility of manipulating discrete multimedia information (text, images) with continuous information (audio, video) [3]. It is reality an Intranet/Internet scenario where the components of a service, the processing objects and the physical devices, located in different nodes, are shared among different applications and users. Conversely, this fact imposes a composition process that is more complex than in the case of stand-alone system [16]. It is required to go further the actual multimedia "microworlds" [3], i.e. self-contained and platform dependent development systems, whose design and evolution is under the control of a single vendor, and that act as a proprietary middleware between applications and low-level system functions. In these situations, processing objects and multimedia physical peripherals can be located on distinct nodes. Proper abstractions are also required for supporting platform independent media devices, and the work carried out in this direction by the Interactive Multimedia Association (IMA) [1] and by the ISO PREMO standard [2] is of the most valuable significance. In this scenario, applications are combined by selecting and plugging-in components from a pool of already developed elements, with no problems of generality and portability conflicts. Additional efforts are also undertaken in this direction, but at a low infrastructural level by OMG [11].

The objective of the MUSIC (MUltimedia Service Composition) project is to devise a multimedia services composition prototype that by means of an intuitive graphical approach, allows to define the main necessary steps for the composition and the customisation of multimedia systems. The distributed platform referenced by MUSIC is OMG CORBA [10], although the processing elements are defined according to the TINA[1] computing architecture [17]. For their definition the TINA ODL (Object

[1] *TINA is an effort carried by a world-wide consortium to overcome current network limitations in the provisioning of a service (http://www.tinac.com).*

Definition Language) [18] has been adopted. The services composed by MUSIC allow dynamic binding at runtime, and make use of platform services (e.g. Trader, Event Service) in order to exchange interface references and for synchronisation activities. The operations among the objects composing a new service are describe by means of a sequence of snapshots, i.e. visual representation of the state of the multimedia flows at a certain time. Proper synchronisation rules apply in order to control the execution of the sequence of snapshots. The composition of a new service is basically made up of the creation and management of a proper sequence of snapshots, composing a final storyboard [8]. A movie-like approach has been adopted, because of its expressiveness, of easy comprehension for the user and because it allows coordination of the "time" dimension. The generated code is a JAVA applet [5], as well as the MUSIC prototype.

2. Related Works

The issue of (distributed) multimedia service composition is relatively new in literature, being many proposals more oriented to standalone multimedia authoring systems [3]. In MUSIC it has been adopted the same approach proposed for the IMA Multimedia System Services (MSS) [1] and being adopted in PREMO [2]: all the control interfaces of all the service components are defined in CORBA IDL. MSS define a synchronisation model of the actions based on the timeline diagrams, very effective but loosely coupled with the graphical composition model. Another interesting research project is CINEMA [2]. It adopts component and nested objects that are conceptually similar to the correspondent entities defined in the TINA computational model [17]. Moreover, in CINEMA a script is eventually generated as a result of the service composition activity. Similar issues are also considered in SAW (Software Assembly Workbench) [7]. SAW is an ATM-based platform for rapid generation of multimedia applications that adopts a graphical model. The expressiveness of the visual programming approach is thereby illustrated, with the possibility of simulating the single service components.

3. MUSIC: the project

One of the main issue in literature is the lack of the adoption for a standard solution for the representation of the objects and for their retrieval, in favour of different proprietary solutions. In addition, nevertheless the usage of graphical formalisms and of visual programming [14] approaches, the definition in temporal diagrams of the object interactions and connection set-ups is not completely tackled. In MUSIC the sequence of events is expressed by means of a proper graphical formalism (the movie-like paradigm, section 4), while the composition issues are coped with embracing well defined and adopted standards: CORBA [10] for the distributed aspects, JAVA [15] for the code mobility and TINA [17] for the definition of the single components and for an overall open architecture. The envisaged user of MUSIC is a business consultant who specifies new multimedia services by assembling existing (processing and device) objects, in order to deploy them on an open system.

3.1 The reference object model

In MUSIC the service components are represented by objects defined according to the TINA computational model [17]. The objects support two types of multiple dynamic interfaces: computational and stream. The computational are simply mapped onto CORBA interfaces, and CORBA IDL is used for their specification; while the stream interfaces are used for abstracting , and for specifying flows of data. They contain a certain number of end-points identifying the multimedia stream flows (sinks and sources).

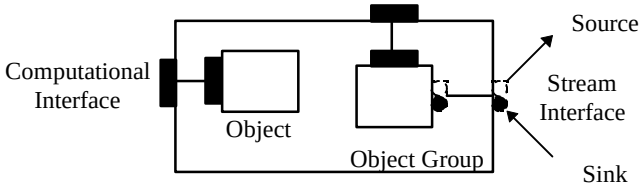

Figure 1: TINA objects and object group.

The objects and their interfaces are described by means of TINA ODL (Object Definition Language) [18] (regarded as an extension of IDL), and proposed to the MUSIC user by means of a graphical representation (Figure 1).

3.2 The basic multimedia components

In MUSIC the TINA (CORBA) objects represent the processing components of a service, whereas the physical devices are depicted as predefined terminal objects. A particular set of icons allows the user to identify the endpoints of a multimedia connection (see Figure 2). They do not intend to be exhaustive, but only placeholders for more precise and detailed hierarchical definitions (we refer to the MSS definitions as in IMA [1] and in PREMO [2]). The concept of port [1], through which a device receives/transmits media data is related to the one of stream endpoint (sink/source) of TINA objects.

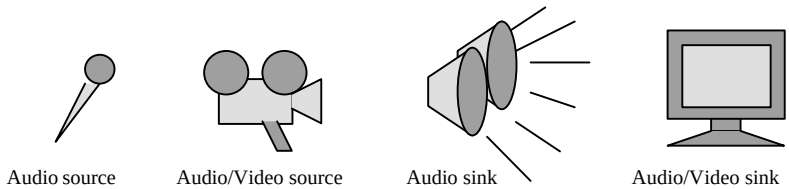

Figure 2: Terminal multimedia objects in MUSIC.

3.3 Code generation

The output of the composition activity is the generation of code that is able: (1) to dynamically retrieve the interface references of each TINA object and for the physical devices located on the network nodes (e.g. by means of the Trader); (2) to execute the operations indicated for the proper customisation and configuration of the service components; (3) to invoke operations offered by a platform API for the setting-up of the connections among the stream end-points of the participating objects. The generated code is a JAVA applet (in order to provide mobility and portability) although it is only a

skeletal file, customisable by the user. The availability of JAVA CORBA brokers (e.g. OrbixWeb [12]) and the availability of CORBA-enabled browsers is therefore a basic factor for devising open distributed multimedia systems.

4. Actions synchronisation by means of snapshots

In MUSIC, the approach for interactively representing the whole composition of a multimedia service is based on the movie-like paradigm [8]. By means of a "storyboard" the user (intended as the business consultant, and not the final user of the service) illustrates all the necessary steps to be executed in order to set-up a new multimedia service.

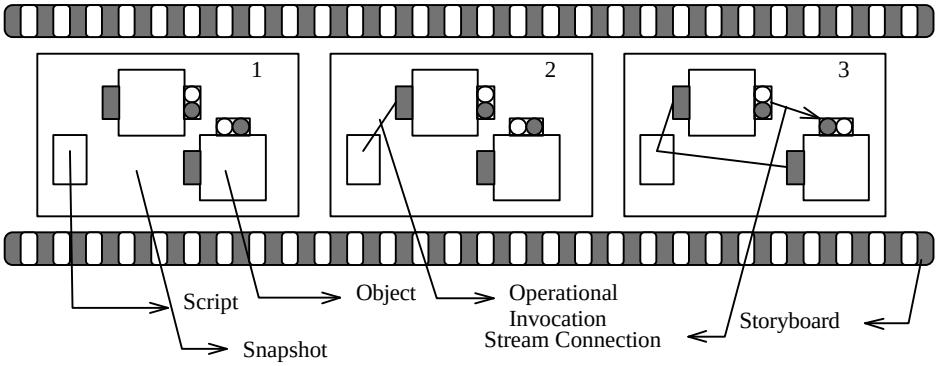

Figure 3: The movie-like composition paradigm in MUSIC.

Eventually, a sequence of snapshots for the new service will indicate the proper phases, e.g. starting from the retrieval of the composing objects, and indicating the proper (operational and stream) dependencies among them.

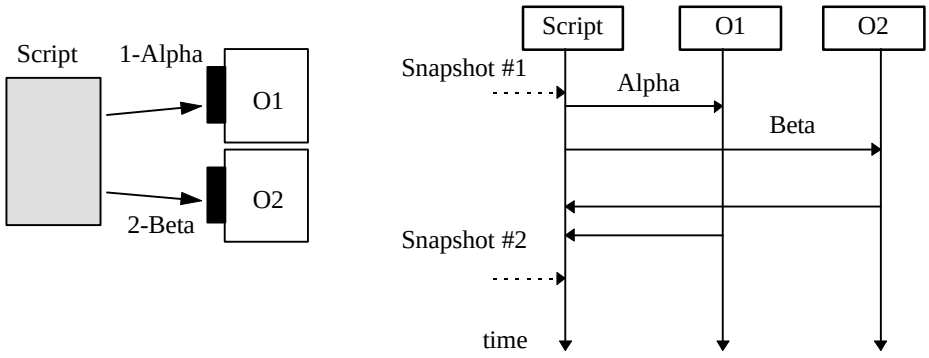

Figure 4: Snapshot #1 (parallel case) and corresponding temporal diagram.

The advantage of such a paradigm relies on the expressive merging of spatial relationships (e.g. representing connections to be set-up) with temporal dependencies (expressed by the sequence of snapshots) occurring among objects. The temporal information is large-grained, regarding only interactions among objects. It doesn't deal

with fine-grained temporal relationships and synchronisations, that are already defined within the processing objects and that are better represented with other approaches (e.g. [4],[5]). The semantics of the underlying adopted model is the following one: all the operations represented within the same snapshot (numbered for the sake of clarity) are intended to be executed in parallel, whenever it's possible(Figure 4). The (n+1)th operation will be activated after the start of the n-th operation, but not necessarily before its completion. In case a strict succession has to be followed (i.e. a certain operation is invoked only after the completion of another one), multiple snapshots must be used. The operations within a snapshot will be started only when all the invocations started in the previous snapshot are completed.

In Figure 4 it is supposed that `Alpha` and `Beta` are two operations whose invocations respectively last 5 and 3 time units. If `Alpha` and `Beta` invocations can start indifferently, they will be indicated in the same snapshot, although the operations will be executed in parallel (in the example,`Alpha` has priority).

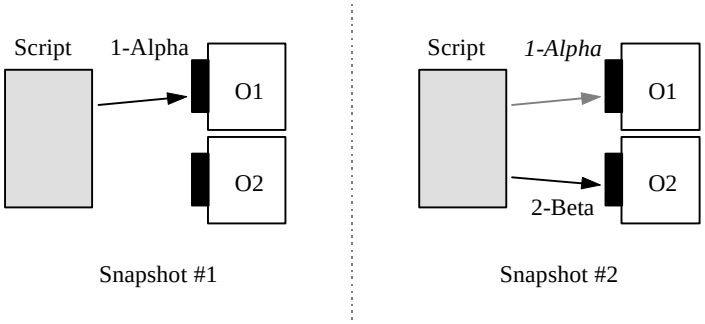

Figure 5: Snapshots #1 and #2 (sequential case).

A different situation can be devised in another scenario (sequential). If the invocation of `Alpha` must be completed before the invocation of `Beta` two snapshots must be used.

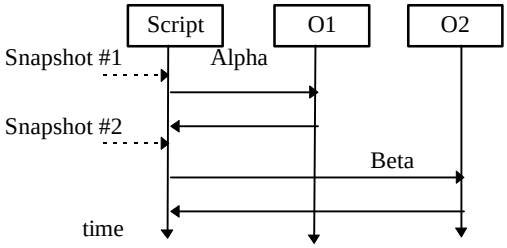

Figure 6: Temporal diagram for snapshots #1 and #2 (sequential case).

From the previous diagrams (Figure 5, where the invocation of `Alpha` is in italics to indicate that it has been previously defined) the Figure 6 temporal diagram can be drawn, where it is highlighted that `Beta` can be invoked only after the completion of `Alpha`. MUSIC offers the possibility of defining, for each snapshot, auxiliary synchronisation conditions, and once that they are satisfied, the invocation of the snapshot can start. These conditions (showed as clock-shaped icons) are expressed

as boolean predicates, referring to local variables declared in a previous snapshots or to any application event. A possible usage is to synchronise the start of the actions of a snapshot with any application generated event (e.g. Start of the multimedia flow). The synchronisation conditions of the initial snapshot are the start-up ones of the service.

5. MUSIC: the composition model

The MUSIC composition model makes intensely use of the features of the visual programming [14]. The components (software modules and devices) are represented by icons and the links among them identify physical connections and object interactions.

5.1 The connections types

Three types of connections are used for modelling object interactions: operation invocations, parameters passing and stream connections.

Operation invocations

This type of connection represents an invocation of an operation offered by a computational interface of an object. The types of invocations have been divided in two additional categories: descriptive and executive invocations.

Descriptive: it's a normal operation invocation between two service components. The user knows of such an interaction by consulting the textual behavioural descriptions for each object (from the ODL specifications) and graphically depicts such kind of interaction for the sake of the comprehension of the service. This kind of representation doesn't generate final code, being the corresponding interaction already defined within the object behaviour. This type of representation is mainly due to the actual lack of a formalism for expressing the basic semantics of an object behaviour.

Executive: it's a normal operation invocation, comprising all the interactions that have to be carried out by the final user in order to properly control the composition of a service. The interface references are retrieved at run time (i.e. when eventually the service is being executed) from the Trader. MUSIC generates in the final script the proper code for these operations. Operation invocations are depicted by continuos arrowhead links, with associated labels identifying the progressive number of the operation invocation and the name of the target interface and operation. The arrow that characterises an executive invocation starts from a particular icon representing the final script.

An example of a simple interaction schema is hereby reported. It is supposed that two objects (A and B) offer an operational interface each one. At the time of the service execution, after having retrieved (e.g. from the Trader) the references to their, it must be invoked the operation "`Start`" of the object A. In turns the operation A invokes the operation "`Service`" of the B object: it is supposed that this behaviour is textually described in the ODL of the object A.

```
Object A {                          Object B {
```

```
  interface i1 {                    interface i2 {
    //operations:                     // operations:
      Start ()                          Service()
       behaviour:  {                  }
       type-Bobj ect.service();    }
       ....
      }
  }
}
```

Figure 7: ODL specifications for class A and B.

In the MUSIC model, the graphical representation corresponding to the situation of Figure 7 is reported in Figure 8, where the invocation of the `Start` operation is an execution, and it is necessary to indicate it because it allows the definition of the service. On the contrary, the invocation of the operation `Service` from A to B is descriptive, and its purpose is to illustrate which operations the object A will execute after having been activated. The generated code contains only the `Start`[2] operation: **`…;i1.Start();…`**

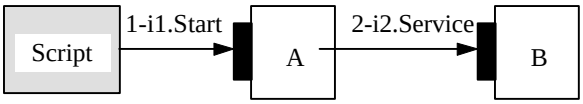

Figure 8: Example of operation invocations.

The code of all the examples is automatically generated by MUSIC: JAVA primitives (OrbixWeb[12]), with invocations of CORBA interfaces.

Parameters passing

The parameters passing completes the scenarios of the possible operational interactions.

Descriptive parameters passing: the user wants to represent that a given reference is passed to another as a parameter (e.g. due to an operation invocation). Anyway, MUSIC doesn't generate code in this case, as this kind of interaction is implicitly defined within the objects' behaviour. The parameters passing is indicated by dotted arrowhead lines. In the example of Figure 9 the MUSIC user has also specified that a reference to a B's interface is being passed as parameter to the object A. It is possible to add textual information to the dotted line, to indicate that the B instance value will be requested at runtime from A (e.g. using a dialog box). The code is: **`...;i1.Start();...`**

Executive parameters passing: it is a special case, when a reference is passed as parameter to an operation invoked from the outside in order to compose the service. The operation invocation is usually executive and also the corresponding parameters passing must be made explicit in the final generated script. This type of parameters passing will therefore produce generated code.

---

2 The code for retrieving the interface offered by A and B will be also produced; for each interface, an Import() invocation will be generated.

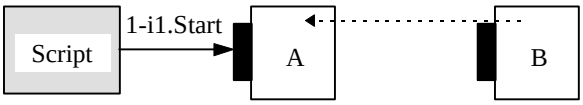

Figure 9: example of generic parameters exchange.

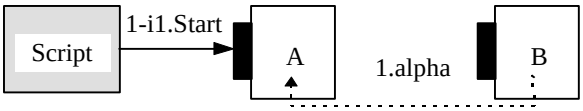

Figure 10: example of parameters exchange in an operation.

The arrow labelled "`1-i1.Start`" indicates that the operation `Start` must be invoked on the interface of type `i1` offered by the object A, while the arrow directed from B towards A (labelled "`1.alpha`") indicates that the reference to B is being passed as the actual parameter "`alpha`" to the operation number 1 invoked on A ("`alpha`" is the name of one of the formal arguments of `Start` of interface `i1`).The code is: **`...;A.Start(B);...`**

Stream connections

The endpoints offered by the stream interfaces may be bound together only if they are compatible (i.e. an MPEG source may be connected only with an MPEG sink, not with an audio source). In MUSIC, the compatibility among endpoints follows the rules of subtyping defined in [17], and this means that two endpoints are compatible when: (1) they have the same type and (2) they have two different types, but subtypes of a common type. The type of an endpoint is obtained from the ODL specifications. A connection between two endpoints doesn't imply that the data stream flows from one object to the other, but only that this can happen at runtime as a consequence of the connections specified by the users. The effective control of the stream (e.g. by means of Start, Stop, operations) is left to the user of the final application. MUSIC, supports only the composition of a new service, and not the service specific details of its usage (that are contained in the processing objects). The code generated for a stream connection is mapped onto an invocation of a proper binding operation. The references of the interfaces are passed as parameters. The binding operation is provided by the connection management service of the underlying platform, responsible for the set-up of a physical connection between endpoints (terminal points of the network) with the proper QoS values.

Figure 11 illustrates an audio data flow starting from a physical source (represented by a microphone icon) and reaching a sink endpoint of a stream interface of object A. This object will process the input flow (e.g. compress the signal) and will generate an output flow that will transit in the connection A-B, that in turn, will do its own processing and eventually the result is transmitted into an audio sink. As indicated, the stream connections may be drawn from sources and sinks of TINA objects, likewise terminal device objects can take part in the connection. The final code for the example of Figure 11 is:

```
StreamBinding(device-mike,objA-name_itf-sink);
StreamBinding(objA-name_itf-source,objBname_itf-sink);
StreamBinding(objB-name_itf-source,device-speakers);
```

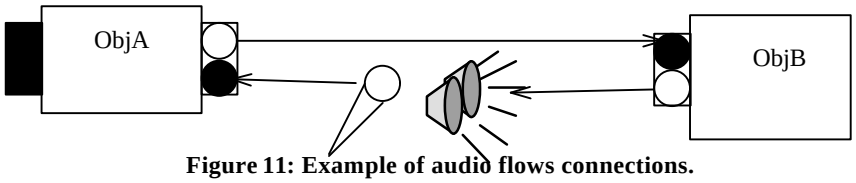

Figure 11: Example of audio flows connections.

The StreamBinding() operation is offered by the Connection Management service of the underlying platform. It is out of the scope of this work to deep into the functional details of a connection management service. The StreamBinding() operation is hereby indicated as an abstraction of the functionalities offered by the platform for setting up proper network connections. It is possible to take into account QoS parameters associated to each stream (extracted from the ODL specifications). The QoS values may be passed to the StreamBinding() function in the generated code.

5.2 The generated code

The user can generate a JAVA applet with skeleton definitions (to be refined) for the methods that are broadly defined in this way:

- "**`init`**": This method (executed by the browser the first time the applet is downloaded) contains the operations (a sequence of `Import()` to the Trader), for the retrieval at run time of the involved objects (processing and physical devices) .
- "**`start`**": In this method (executed each time that the page containing the applet is exposed) all the operations related to the connections among objects are reported, and in particular: (1) invoking operations on interfaces; (2) expressing the actual values of the formal parameters passed to the invoked operations; (3) invoking the Connection Management `StreamBinding()` operation (5.1). Moreover, for each synchronisation condition, a proper waiting instruction is generated.
- "**`stop`**", "**`destroy`**": They are not generated, although the user can add any proper additional operations related to the termination of the service.

6. Future Works and Conclusions

The illustrated concepts and ideas constitute part of our current activity, and a first prototype (integrated with OrbixWeb[12]) has been produced, although some additional work is still needed. Next evolution will comprise a tight integration with tools supporting behavioural specifications (e.g. ACE [9]), as well the usage of platform services (e.g. Event Service).

7. References

[1]IMA, *Multimedia System Services*, Draft Recommended Practice, 1995

[2]ISO, *Presentation Environments for Multimedia Objects - Part 1*, 1996

[3]M.Mühlhäuser et al., *Services, Frameworks, and Paradigms for Distributed Multimedia Applications*, IEEE Multimedia, Fall 1996

[4]S.Gibbs, *Composite Multimedia and Active Objects*, OOPSLA'91, 1991

[5]S.Gibbs et a., *Multimedia Programming Objects*, Addison-Wesley, 1994

[6]K.Rothermel, I.Barth, T.Helbig, *CINEMA-An Architecture for Configurable Distributed Multimedia Applications*, Universität Stuttgart, 1994

[7]E.R.Beyler et al., *An ATM-Based Platform for Rapid Generation of Multimedia Applications*, AT&T Technical Journal, 1995

[8]P.G.Bosco, *Requirements for Multimedia Service Creation, CSELT*, 1996

[9]P.G.Bosco, G.Martini, C.Moiso, *ACE: An environment for specifying, developing and generating TINA services*, IEEE/IM'97, May 1997

[10]OMG, *The Common Object Request Broker: Architecture and Specification*, 1995

[11]OMG, *Control and Management of A/V Streams RFP*, OMG n. 96-08-01

[12]IONA, *OrbixWeb Programming Guide*, 1996

[13]R. Orfali et al., *The Essential Distributed Objects*, Wiley, 1996

[14]M.M.Burnett et al., *Visual Object-Oriented Programming*, Manning, 1995

[15]J.Gosling, H.Mc Gilton, *The JAVA Language Environment*, Sun, 1996

[16]TINA-C, *Service Composition Problem Statement*, 1996

[17]TINA-C, *Computational Modelling Concepts, vers.* 3.2, 1996

[18]TINA-C, *Object Definition Language Manual*, 1996