BT position paper - The use of Reflective techniques in middleware for end to end QoS control of real time applications on the Internet

J. Tassel

Applied Research and Technology
BT Laboratories, UK
Martlesham Heath, IP53RE

Abstract

This position paper describes some of the work done and forthcoming at BT in the area of reflective techniques to support real-time services. Reflection provides flexibility, adaptability, system independence and dynamic facilities and is therefore a very suitable technique to provide off-the-shelf flexible software components for real time middleware. A current result of the work in this area is a component using Reflective Java which provides QoS control for multimedia Java Internet applications by using resource reservation techniques. This component controls the sharing of network and host resource and enables the user to express his preferences regarding the spending of his budget.

1. Introduction

Current work at BT is focused upon building a middleware platform to support real-time delivery of multimedia information in a large scale commercial multicast environment. An inherent property of the Internet is that applications and users have evolving requirements and that the communication environment changes with time. There is therefore not a unique solution to the problems related to real-time delivery of data. Protocol flexibility must be integrated into the middleware platform to support heterogeneity, evolution and adaptation to the environment on the Internet. We are looking at the use of Reflective techniques to provide flexible components in our middleware which will solve this important technical issue.

Reflection is a software engineering technique which separates non functional concerns (real-time delivery of data) from the application core objective (sending data) by using the Meta Object Protocol (MOP). The MOP binds (at build and run time) application objects to non functional objects (called meta objects) by using a reflection object (third party binding object). Meta objects are written independently of any applications and permit the interception of invocations of application object methods so that meta object code can be executed before and/or after those methods. The insights of the MOP are presented in appendices for readers who are unfamiliar with Reflective techniques.

The use of Reflective components provides:

- **Policy and mechanism independence:** Application objects provide the functional behaviour of the application and meta objects (which are application independent) provide the non functional requirements (QoS control, security, replication, etc.). In the implementation both types of objects are unaware of each other.

- **Dynamic facilities:** Meta objects can be bound to an application at build and run time so that, as the environment or requirements change, a more appropriate meta object can be used.

- **Adaptation**: by using dynamic facilities the application can adapt its behaviour by replacing meta objects at run time. A meta object can also monitor the application and trigger the adaptation.

- **Easy upgrade, flexibility**: as meta objects are application independent they can be upgraded independently. The same applies to the application.

In order to provide real time services over the Internet, the QoS of the data communications must be controlled. This control is a non functional aspect of the application that can be implemented using Reflective Techniques and the Meta Object Protocol. Reflection is therefore ideally suited to build off-the-shelf flexible components to provide real-time control. The following sections of this paper describe the technical interest of BT in Reflection to build real-time applications and details some of the work in this area.

2. The use of Reflection in QoS control for real-time services

The traditional approach to add QoS control to an application is to use a QoS management API (Application Programming Interface). Using an API means having to understand the application code and modifying it in order to use the API functionality. This also means adding complexity to the application, reducing the flexibility and complicating later upgrades of the application. We believe Reflection is an efficient technique to solve these issues. The commercial and strategic background to the interest in BT for this work is the desire to encourage best practice in software engineers from supplier companies. We are doing this now, so that the QoS controllers they produce are easy to integrate with existing and forthcoming applications, are easier to debug, can be written by specialists and their time to market is reduced. It is the best time for this work as QoS control for the Internet is about to be delivered and therefore we can encourage best practice in software engineers as they are starting to think about adding QoS control in their applications. For example, a joint venture between CISCO/MCI/Intel has produced a set of RSVP enabled applications [1]. One of the problems they encountered during this work was the amount of time required to modify the existing applications (one person a month per application). The problems they had regarding the presentation of the QoS control to the user were repeated for each application.

3. Work at BT on real-time control and the use of Reflective techniques

3.1 QoteS - Quality of Service around the Edge

We are working on a component using Reflection for real-time control: QoteS [2]. QoteS is a QoS management component for real-time Internet multimedia applications (receiver side only) which applies the pre and post method interception facilities of ANSA's reflective Java (cf. Appendix A) to do some QoS related work when a non-QoS aware application opens a Java socket. When the application is written it does not need to be aware of QoteS and QoteS is independent of any applications which might use it. Socket creation calls later need to be modified so that a modified Socket class is used. This modified Socket class inherits from the original Socket class and uses QoteS via a QoS control meta object. As a result the original application behaviour is retained and QoS control is added.

The QoS related work is primarily concerned with allocating and sharing resources in the end system and network. Network resource control is provided by using the IETF RSVP and a trivial host admission control algorithm is used to control the CPU, memory and hard drive usage. RSVP messages are not consumed by the application but by QoteS where a QoS manager processes them to set charging and to reserve both host and network resources.

To drive the allocation of resources, there is a GUI through which the user can state general QoS preferences (such as priorities across different applications and different streams) and a simple economic model that constrains resource usage. The user sets his budget and by using this GUI defines how he wants this budget to be spent over his applications and streams, modifying priorities between streams, modify their running cost and the amount of resources they are allocated. This work has highlighted many issues related to charging on the Internet for network resources, which is the subject of further work.

The following is a snapshot of this user GUI:

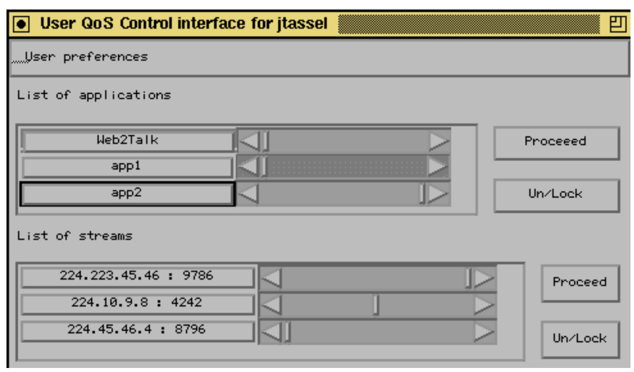

Fig. 1, User GUI

Reflection in QoteS enables this component to be very easily added to any Java Internet application which uses the java.net.Socket class. We added it successfully, in half a day, to a one-to-one telephony application we randomly chose on the Internet. QoteS includes a run-time switching mechanisms for adaptation to a changing environment. The meta object used to call QoteS in the Socket creation call can be modified at run time so that it calls a different QoS control architecture better adapted to the new applications requirements (i.e: moving from a fixed to a mobile network)

More details about QoteS can be found in [2] and in [3]. Those two papers present the design and implementation of the QoteS prototype and [3] also discuss the limitations of Reflective Java.

3.2 LSCMA, QoS control

The long term project the author is involved with aims at providing a "Component based middleware for Internet real time large scale commercial multicast applications (LSCMA)". The work is being focused on the scalability issues of the security, charging, QoS control, data handling, service discovery and session control protocols.

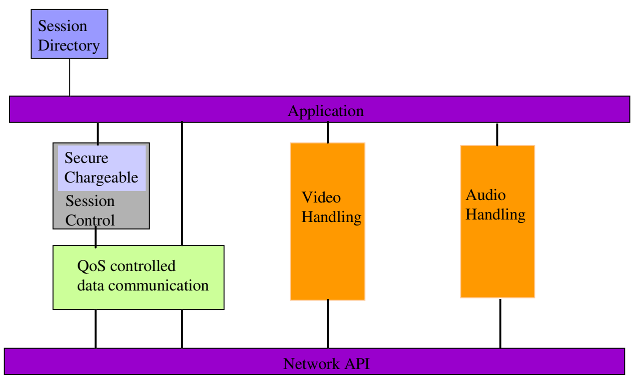

Fig. 2, Middleware for Real Time transport of data

An important aspect of this middleware is the QoS control to deliver real-time services. Reflection is being investigated as one option to provide flexibility and adaptation. Layered multicast is one option being explored for delivery of different quality of service. A background paper for this work can be found in [4].

3.3 ANSA project: Flexinet

FlexiNet [5] considers the separation of application development and deployment. Particularly concentrating on deployment of the communication stack. By providing a specification of the requirements of an application, and a description of the environment in which it is to be deployed, FlexiNet will develop a framework which will mask deployment issues ensuring that all parts of the application respect the agreed policies, at all times, in all circumstances.

There are three trends in system design which strongly support the need for FlexiNet. Component architectures are increasing in popularity. By building applications from components, not only is the development time reduced, but there is an increased freedom with the choice and placement of components. For example alternative components may be used to provide the same function when deployed in different circumstances and with different QoS requirements: real-time or not.

A second trend is in introspection, this is the provision of information about the running system and this is where Reflective techniques are used. This gives an application the ability to examine its environment and act appropriately. For example, if we can determine the topology of a network, we can assess the most efficient way to distribute an application.

The third trend is business needs driving application development. Modelling the need provides us with the information about the business goals and constraints, and allows us, to determine the best approach to meeting these constraints in a particular deployment. For example we may adopt different tactics validating and auditing communication between parties in the same organisation and between those in different organisations.

3.4 Adaptive reflective middleware for distributed multimedia applications

Another project investigating the use of Reflection, in which BT is involved, is taking place at Lancaster University, UK. This project [6] is looking at providing Reflective middleware in distributed object architectures (CORBA), to increase flexibility and openness and support real-time multimedia services.

BT is also interested in investigating the use of Reflective techniques to provide a real-time ORB to support real-time multimedia services and in a more general case to provide non-functional services in ORBs. Another area of interest is the study of the cost/performance balance of using Reflection to provide real-time services.

Appendix A. (adapted from section 2.3.2 in [3])

Introduction to Reflective techniques and Reflective Java

We present Reflection in the following paragraph by means of Reflective Java but Reflection has broader properties than the ones implemented in Reflective Java [6].

Reflective Java is a product from the ANSA research consortium [7]. It allows the separation of the functional concerns of an application from the non functional ones. This is done using the Meta-object Protocol (MOP) [8]. The MOP separates the roles of application developers, one focuses on what the application does and an other on how it does it. One can imagine some requirements necessitating a particular implementation in a fixed network environment and an other in a mobile network environment, Reflective Java permits the dynamic update of the implementation of these requirements as the environment changes. The following diagram illustrates how Reflective Java implements the Meta-object Protocol :

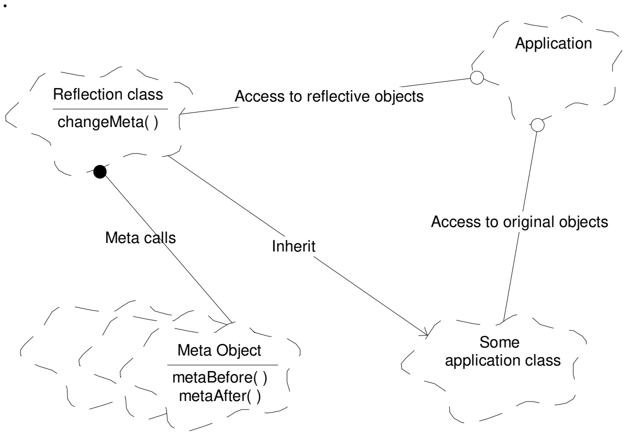

Figure 1, The MOP implementation in Reflective JAVA

In an object oriented world, an application is made up of classes. On the right side of the model above we can see that an application uses the services of some other class. Reflective Java allows to upgrade the service of classes and still retain their original behaviour. Methods in the original class are said to be made reflective by binding them to some extra code to be executed before and/or after their invocation. The reflection class on the diagram above acts as a third party binding entity (a stub) between the original class and the newly defined extra behaviour in the meta-object class.

We now define the different elements which make-up reflection. The role of the developer of meta-objects is to design and implement the meta-object which defines some non functional behaviour. Examples are meta-objects implementing the well known transparencies of the distributed systems world: concurrency control, replication/location, persistency, failure ...etc. A main feature of the meta-objects is that they are designed and implemented independently of the application context in which they will be later used. Meta-objects are written once and can be used by many applications later. Moreover the meta-object is bound dynamically to the reflection class so that it can be changed at run time, a new version of a service can be installed at run time without the need for modifying the application code.

The meta-object defines a couple of methods, “metaBefore” and “metaAfter”. The code which implements the required extra behaviour is implemented in those two methods. The extra behaviour is, in fact, what should be done before and/or after the “normal” application method execution.

The following diagram illustrates this in the context of controlling QoS over a network socket connection establishment :

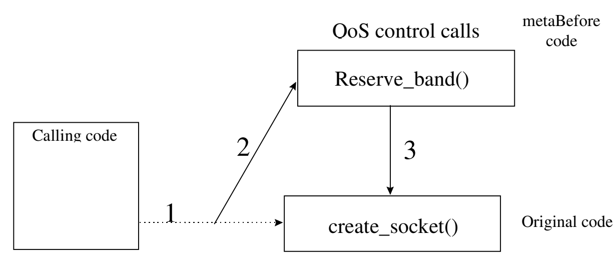

Figure 2, Meta-object processing in Reflective Java

The calling code invokes the create_socket method, but with Reflection, the metaBefore method in the meta-object is invoked first (2) and then the create_socket method is invoked (3) through inheritance. In this example, the metaBefore method in the meta-object makes the reservation of bandwidth and then the socket is created. metaBefore is called before the invocation of the application class original method and metaAfter after the invocation. These two methods have access to the parameters of the original method calls. The code in these two methods is independent of the class which will use them later so that the meta-object can be used for any application.

Reflection also provides some more flexibility by allowing the programmer to make only some of the application class methods reflective. A script language is used to specify statically which meta-object is associated with which application class and which methods of the original class are to be made reflective. As we stated earlier the meta-object associated with an application class can be dynamically modified. This script language is as follow:

```
refl_class      original_class      :
meta_qosrsvp{
  public original_class(): 1;
  public  void  create_Socket(int  ip,
int port): 101;
  public   void   release_Socket(int
handle): 102;
}
```

The refl_class keyword indicates that the original_class is to be made reflective and that it will use the extra behaviour defined in the meta_qosrsvp meta-object. The numbers appearing on the right side of the method signature are category identifiers (these are explained later). It is the role of the system integrator, who has knowledge about the application and the meta-objects to decide what extra behaviour is required for which of the original methods and to write the appropriate script file.

The identifier numbers are used to create different sections of code in the meta-object. The identifiers are used in the script file to specify which section of the code is to be run for which application method. An example is given bellow :

```
metaBefore(param, category_id){
  if(category_id==101)
          reserve_bandwidth(…);
  …
  }

  metaAfter(param, category_id){

if(category_id==102)

  release_reserved_bandwidth(…);
  …
}
```

With the example meta-object and the script file example above, the original create_Socket() method will cause the establishment of some bandwidth reservation using RSVP before the socket is created and the communication handle returned. The release of a socket will cause the reservation to be released after the socket is freed. The code in the application only deals with socket programming and the meta-object only with QoS control. The script file is used to bind the two together. Finally, the Reflective Java pre-processor is run over the script file and produces the reflection class (cf. figure 1), which implements the same methods as the original class wrapped with the extra code provided by the meta-object. We recall that the reflection class inherits the behaviour of the original application class. The application then needs to be updated manually by the system integrator to instantiate socket objects which require QoS control from the reflection class and not the original socket class. This pre-processor is written in C++ and only runs on Solaris and HP-UX. It produces Java code which can then be used on any operating system, so the operating system limitation is minor.

Other features of Java such as network download of classes enable this scheme to be very flexible.

The main difference between components using Reflection and the traditional view of components (Java Beans for example) is that the application does not make explicit calls to them. We believe that as real-time software components provide a non-functional service, Reflection is a very appropriate mechanism to integrate them with applications (because of the flexibility they provide). Whether they are integrated directly with the applications or are part of the middleware the applications uses.

References

[1] Commercial Internet Multimedia Trials - Preliminary results - Fred Baker, Cisco; Mark Baugher, Intel; Saib Jarrar, MCI; Henry Sinnreich, MCI. Munich IETF, RSVP working group. Set of slides. ftp://ftp.isi.edu/rsvp/ietf.meetings/munich.ietf/

[2] A price based end to end QoS control component for Internet multimedia Applications using Reflective Java. Jérôme Tassel, Bob Briscoe, Alan Smith. BT, Advanced Research and Technology. To be published at the 97' COST237 conference in Portugal. http://www.labs.bt.com/people/briscorj/projects/lsma/cost237.doc

[3] Quality of Service (QoS) adaptation using Reflective Java, Jérôme TASSEL. A dissertation submitted for the degree of M.Sc. in Distributed Systems Computing Laboratory, University of Kent at Canterbury, 19th September 1997. Project funded by British Telecommunications plc. Contains an extensive bibliography.

[4] Real time applications on the Internet. Steve Rudkin, Andrew Grace, Mike Whybray. BT Labs UK. BT Technology Journal vol 15 no. 2 April 1997.

[5] Flexinet, Automating application deployment and evolution. http://www.ansa.co.uk/Research/Flexinet.htm. Martin Robertson.

[6] Notes on reflective middleware, Gordon S. Blair. Department of Computing, Lancaster University. (to be published)

[7] Design of Reflective Java. ANSA, Zhixue Wue and Scarlet Schwiderski, 23rd December 1996. APM.1818.00.06 (22 pages) http://www.ansa.co.uk/Research/ReflectiveJava.htm. Public copy at ftp://ftp.ansa.co.uk/phase3-doc-root/APM.1936.02.ps.gz

[8] Computational Reflection in Class based Object Oriented Languages. Jacques Ferber, LAFORIA. Object-Oriented Programming, Systems, Languages and Applications (OOPSLA) Proceedings, 1-6 October 1989. (8 pages)