New Object Models for Seamless Transition Across Heterogeneous Mobile Environments

Anjaneyulu Pasala **D. Janaki Ram**
Distributed & Object Systems Group
Department of Computer Science and Engineering
Indian Institute of Technology, Madras, India
E-mail:{anji,djram}@lotus.iitm.ernet.in

Abstract

In mobile collaborative applications, collaborators may move across heterogeneous environments. This paper proposes object models for seamless transition of collaborators across heterogeneous environments. This is achieved by framing the various limitations of the environment as constraints and abstracting them into a constraint meta-object. *Collaborator objects are dynamically customized by attaching the appropriate* constraint meta-object *based on the environment in which the collaborators are working. Remote customization of objects is achieved by 'distributed glue model'.*

Key words: Object Models, Heterogeneous Environments, Mobile Applications, Adaptation

1: Introduction

Wireless communication and mobile computing make it easy for people to collaborate irrespective of their physical location. Flexible collaboration in mobile environments means that applications should not be based on a single type of network [1, 4]. These applications must adapt to high speed wired networks, low speed wireless networks and dial-up lines and the selection depends on the current physical location of the user. These networks have varied characteristics such as different data rates, data error rates, QoS parameters, latencies and usage costs. Moving across these networks calls for imposing some limitations for efficiency reasons. For example, when a user moves from a high speed network to a typically low bandwidth wireless environment, he has to forego some of the special features like color due to the low bandwidth of the wireless connection. Other limitations that the user may come across in such heterogeneous environments are the assumptions that the user makes about the facilities available with other collaborators such as multimedia equipment, graphics display, etc. Some may be having multimedia tools while others may not. Hence the application must be able to dynamically adapt itself to these changing environments.

This paper proposes a *Mobile Constraint Meta-Object* (Mobile-CO) model that separates the environmental limitations from the actual functions of the collaborator. The environmental limitations are abstracted into a *meta-object*. Depending on the collaborator's current working environment the corresponding *meta-object* is attached to the collaborator object at run-time. Thus the separation of environmental limitations from collaborator objects provides a flexible adaptation to different network environments. It also provides extensibility. A distributed glue model is proposed to remotely customize the objects at run-time.

2: The Mobile Constraint Meta-Object Model

It is possible to achieve flexible collaboration in heterogeneous environments by framing various limitations of an environment as constraints and abstracting these constraints into a *meta-object*. For example, wireless environment cannot effectively transfer color images. In case the data is in color, the color should be suppressed before transmitting the data. This limitation can be framed as a constraint. The constraint is validated before the data being forwarded. Similarly other limitations of the environment can be framed as constraints. The data that get transmitted should satisfy all the constraints specified in the *meta-object*. Depending on the environment in which the user is currently working in, the corresponding *meta-object* can be dynamically attached to the collaborator object. Appropriate *meta-objects* are attached and detached upon switching of networks. Since the *meta-object* encapsulates the constraints of an environment, it is named as a *constraint meta-object* [3]. The *constraint meta-object* captures the messages sent to the collaborator object and validates them against its constraints. Hence, the behavior of an object depends on the currently attached *constraint meta-object*.

The *constraint meta-object* model is based on a clear separation of the functionality of a collaborator object from its environmental constraints. The *constraint meta-object* consists of a set of constraints that capture the internal state of the environment with which it is associated. *Constraint meta-object* can be dynamically plugged and unplugged to the collaborator object using `plug` and `unplug` operators respectively. When a new constraint environment is created due to technological developments, a new *constraint meta-object* can be created and attached to the collaborator object when a collaborator enters into that environment. Hence it provides extensibility. The *constraint meta-objects* are instances of the constraint meta class. The constraint meta class can be specified as follows:

```
class name_1 : constraint name_2 {
 <label1>:<constraint>:<semantic procedure> //constraint specification  ...}
```

where name_1 refers to the constraint meta class name and name_2 refers to the class to whose instance this constraint meta class instance is attached. The same constraint meta class can be a constraint class to more than one class. In that case, commas are used to separate the classes. That is, an instance of a constraint meta class can be attached to more than one class instance. The constraint definition in a constraint meta class consists of two slots. One slot is the constraint specification and the other slot is the semantic procedure that is executed when this constraint is violated. The *constraint meta-object* can be made to govern the internal state of the object to which it is attached. As the collaborator object has no information about the environment in which the collaborator is working, an environment object is introduced that will hold the state of the environment. A reference is made by the *constraint meta-object* to the *environment object* to know the state of the collaborator's environment. That is, the constraints of the *constraint meta-object* capture the state of the *environment object* rather than the collaborator object to which it is attached. This arrangement is known as *Mobile Constraint Meta-Object* (Mobile-CO) model. The Mobile-CO model is depicted in Figure 1.

To achieve flexible and seamless collaboration in mobile environment, we have identified four types of constraints. The purpose of each of these constraints is explained below.

1. Receiver's Constraints at the Receiver (RCR): The *constraint meta-object* is attached to the collaborator object. The client messages travel to the location of the collaborator object and are intercepted and validated by the *constraint meta-object* before being delivered to the collaborator object. These constraints are called receiver based constraints,

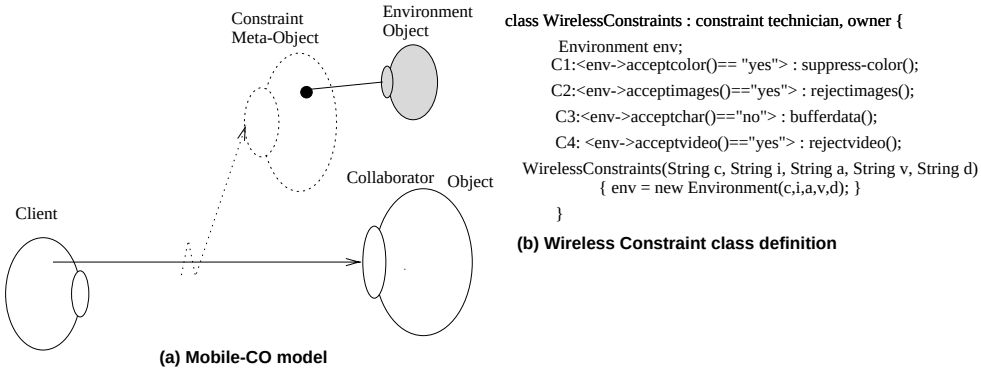

Figure 1. Representation of Mobile-CO model

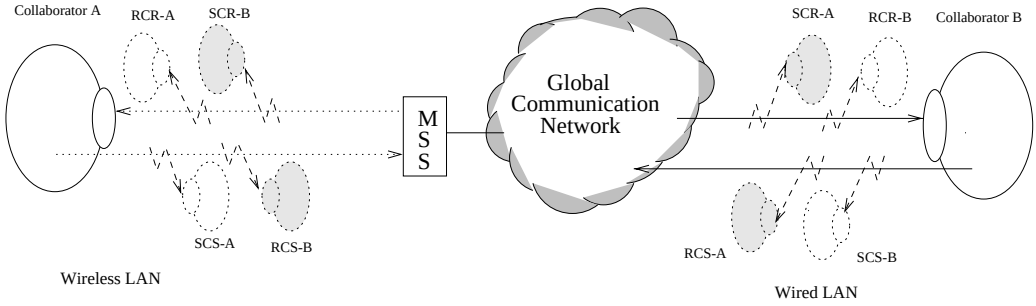

Figure 2. Different types of Constraint meta-objects in mobile environments

because when the messages are received at the receiver, the *constraint meta-object* captures the messages and validates them. Hence the user plugs the constraints as demanded by the environment in which the user is currently working in. These type of constraints are called as Receiver's Constraints at the Receiver. For example, collaborator A working in wireless environment, as shown in Figure 2, plugs the wireless constraint meta-object so that the messages sent to collaborator A are intercepted by the RCR-A and are validated.

2. Receiver's Constraints at the Sender (RCS): Consider the following situation. Assume that collaborator A is working on a mobile computer using wireless connection and collaborator B is working on a workstation connected to high speed LAN. Both the collaborators have plugged their respective *constraint meta-objects* (that is RCR constraints), namely, wireless constraint meta-object and wired constraint meta-object. Further assume that the workstation of collaborator B does not have multimedia tools at present. In case the collaborator A sends video frames, these frames travel all the way to the workstation and get suppressed by the *constraint meta-object*, RCR-B. It does not save the bandwidth of wireless communication which is scarce and costly. The bandwidth and time can be saved if the validation of messages takes place at the sender's location itself. These constraints are named as Receiver's Constraints at the Sender because the constraints put by the receiver are validated at the sender. The RCS constraints are basically RCR constraints at the sender. This can be achieved by bringing the RCR *constraint meta-object* of the receiving object to the sender's location, as shown in Figure 2.

3. Sender's Constraints at the Sender (SCS): In a wireless environment, the user not only has limitations on receiving the data but may also impose restrictions like compress the data and suppress color, on the data to be sent. These limitations are imposed to

save bandwidth and power. These limitations are abstracted as constraints. These are the constraints which are applied on the outgoing messages of an object. That is, whenever an object generates (internally) messages to another object, these messages have to be validated. For instance, consider an example where a collaborator is working in a wireless environment. The user wishes to minimize the sending of data as much as possible. Therefore whatever data the object sends should be minimized by removing the color and further by compressing it, irrespective of the receiving object's constraints. In such cases, all the out going messages should be passed through the constraints imposed by the object which generates them. These types of constraints are termed as Sender's Constraints at the Sender.

4. Sender's Constraints at the Receiver (SCR): The need for these types of constraints arises because of SCS constraints. In the above example the user uses a compression algorithm to compress the data before it is sent. To make the constraints fully transparent to the receiver, the data should be uncompressed before being delivered to the receiver. This type of arrangement is named as Sender's Constraints at the Receiver. The functions of the RCR constraints are reciprocal to the SCS constraints. This type of arrangement facilitates the collaborators to use their own data compression and decompression techniques.

3: Implementation of Constraint Meta-Objects using Distributed Glues

The *constraint meta-objects* have been implemented by using distributed glues developed based on the glue model [2] proposed for object reuse by customization in object-oriented systems. The glue model facilitates dynamic customization of object behavior. An object consists of a fixed or basic behavior and variable behavior. A variable behavior can be changed by attaching a new variable behavior at run-time. The basic behavior and the variant behavior of an object are defined in two separate classes. The class that defines the basic behavior is called the base class and the class that defines the variant behavior is called the glue class. The mechanism of Type-hole is used to specify the variant behavior of a class. A Type-hole consists of a set of method declarations that are specified as part of a base class and definitions are provided in a glue class. This mechanism helps in enhancing and modifying the behavior of a base class by providing alternative or different definitions for the base class Type-hole methods in glue classes.

The glue model proposes four ways of specifying relationships between base and glue objects. These are *part-of*, *using*, *in* and *out* relationships. Both *part-of* and *using* relationships allow the instances of base class to use the instances of glue classes. The *part-of* relationship provides access to the data part of the base class. The *in* and *out* Type-hole relationships allow the messages of an instance of a base class to be intercepted and manipulated by the glue objects. The in-glue objects intercept and manipulate the messages sent to its base objects. Whereas the out-glue objects intercept and manipulate the messages that emerge from the base objects.

As explained in the previous section, the basic function of the *constraint meta-object* is to dynamically intercept and validate the messages sent to the collaborator objects. Hence, the *constraint meta-objects* can be elegantly modeled with the *in* and *out* Type-hole relationships. The RCR and SCR *Constraint meta-objects* have *in* Type-hole relationship with the collaborator objects. Whereas SCS and RCS *constraint meta-objects* have *out* Type-hole relationship with the collaborator objects.

The dynamic customization is achieved by attaching in-glue and out-glue objects to the collaborator. For example, consider the situation depicted in Figure 2 where collaborator A

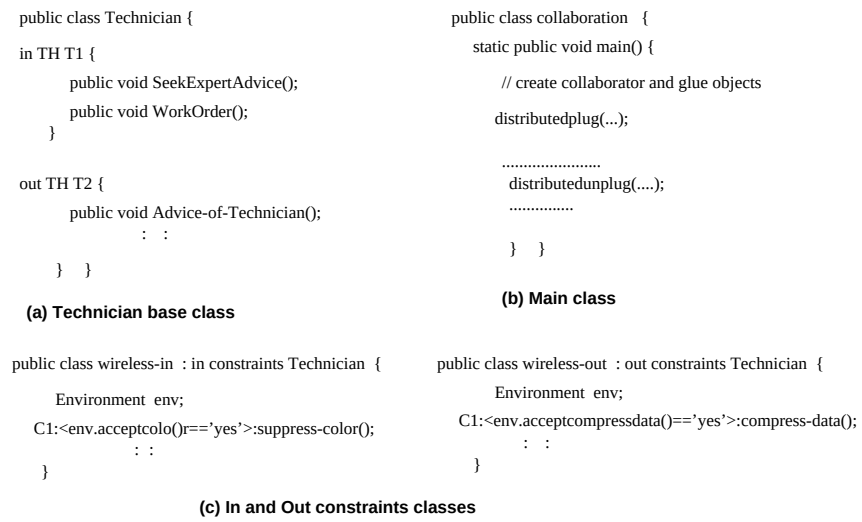

Figure 3. Sample Base and Glue classes

is operating in a wireless network environment and collaborator B is operating in a wired network environment. Wireless-RCR and wireless-SCS constraints are to be attached to the collaborator A to intercept its in-coming and out-going messages respectively. Similarly, wired-RCR and Wired-SCS constraints are attached to the collaborator B to intercept its in-coming and out-going messages respectively. These *constraint meta-objects* can be attached to the collaborator objects using the `plug` operator.

Similarly, the wireless-SCR and wireless-RCS constraints have to be attached to the collaborator B, and wired-SCR and wired-RCS constraints have to be attached to the collaborator A. While the decision to attach these constraints is made at one node, the actual attachment is made to another object residing at another node. Hence, a need for remote customization of objects exists. To achieve remote customization of objects two language constructs, namely, `distributed_plug` and `distributed_unplug` have been provided. The `distributed_plug` operator is used to glue the two objects at the initiator node and two objects at remote node whose classes are in glue relationships. The `distributed_unplug` operator is used to terminate the glue binding between the base objects and their glue objects.

In distributed glue model, a new object called distributed-glue manager has been introduced for gluing of objects both locally and remotely. The main function of the distributed-glue manager is to attach the glue objects on receiving the `distributed_plug` message and detach the glue objects on receiving the `distributed_unplug` message. On receiving the `distributed_plug` message the distributed-glue manager attaches locally the glue object to the collaborator object and sends a message called *localplug()* to the distributed-plug manager at the other node. On receiving the *localplug()* message it attaches the glue object to the collaborator object. The corresponding glue object has to be migrated to the other node. These operations are automatically performed by the distributed-glue manager objects.

Figure 3a displays a code segment with Type-holes T1 and T2 declared as part of a class Technician. This class is known as base class. The keyword **TH** signifies that any class that provides a definition for the method *SeekExpertAdvice()* can aid the class Technician in completing its behavioral specification. A base class is organized like any other class except the addition of the keyword **TH** which qualifies for the Type-hole and a name for the Type-hole. Figure 3a shows that there could be any number of Type-holes declared in a class.

Method names in the base class should be unique.

Figure 3c displays a code segment for *in* and *out* constraint classes. These classes are known as glue classes. The presence of `in constraints` after the class name (in the definition of a class) signifies that the constraints defined in a class are applied on all the messages invoked on the *in* Type-hole methods of a class Technician. Similarly, the presence of `out constraints` after a class name signifies that the constraints defined in the class are applied on all the messages that are being sent from the *out* Type-hole methods of a class Technician.

Implementation of Distributed Glues: The distributed gluing scheme has been implemented by extending the Java language with language constructs proposed in the scheme. A parser is written in Perl which takes glue code and constraints code as input and outputs Java code. Each Type-hole specification in a base class is parsed into an interface declaration. A class which is in glue with a base class implements the corresponding interface. Every Type-hole in a base class is replaced with a reference to the interface represented by the Type-hole. Every message sent on an *in* Type-hole method is first sent to the object's interface reference. Every message that is being sent from the *out* Type-hole method is first sent to the object's interface reference, i.e. the message is forwarded to the currently plugged glue object. The parser generates the DistributedGlueManager class with appropriate functions. The `distributed_plug` and `distributed_unplug` statements are replaced with a message invocation on the distributed-glue manager object.

4: Conclusions

A new object model called *Mobile Constraint meta-object* has been proposed to efficiently handle the heterogeneity present due to multiple network services and architectures. The model frames the environmental limitations as constraints and these constraints are abstracted into a *constraint meta-object*. Programming language constructs have been proposed to dynamically attach and detach the *constraint meta-object* to the object. Four types of *constraint meta-objects* have been identified to make the constraints truly transparent to the user objects. To remotely customize these objects a distributed glue model has been proposed. Though these *constraint meta-objects* introduce some overhead, the advantages are significant in terms of the flexibility and extensibility they provide. The models also facilitate component oriented programming.

References

[1] Bernd Bruegge and Ben Bennington, *Applications of Mobile Computing and Communication*, IEEE Personal Communications, February 1996, pp 64–71.

[2] Janaki Ram D et. al., *The Glue Model for Reuse by Customization in Object-Oriented Systems*, TR No. IITM-CSE-DOS-98-02, CSE Dept, Indian Institute of Technology, Madras, India.

[3] Janak Ram D et. al., *Constraint Meta-Object: A New Model for Distributed Collaborative Designing*, IEEE Trans. on SMC, Vol. 27, Part A, Issue 2, March 1997, pp. 208–221.

[4] Nigel Davies et. al., *Supporting Collaborative Applications in a Heterogeneous Mobile Environment*, Computer Communications, Vol. 19, No. 4, April 1996, pp. 346–358.