Reflective Architectures for Adaptive Information Systems

Andrea Maurino, Stefano Modafferi, and Barbara Pernici

Politecnico di Milano,
Dipartimento di Elettronica e Informazione,
Piazza Leonardo da Vinci, 20133 Milano, Italy
{maurino, modafferi, pernici}@elet.polimi.it

Abstract. Nowadays the anytime/anywhere/anyone paradigm is becoming very important and new applications are being developed in many contexts. The possibility of using applications along a wide range of devices, networks, and protocols raises new problems related to delivery of services. Current academic and industrial solutions try to adapt services to the specific distribution channel, mainly by changing the presentation of the service. In this paper, we reverse this perspective by using adaptive strategies to try to adapt the delivery channel to services as well. We present a possible architecture and focus our attention on the use of reflective components in the adaptive process. Using the reflection principle, we are able to evaluate the channel constraints and the conditions in which the distribution channel is working at a specific time. This information, built with service, user, and context constraints, is used as input to adaptive strategies to change the current channel characteristics, to new ones satisfying all the requirements. If this kind of adaptation is not possible, we consider the different QoS levels offered by the service and the user's readiness to accept a downgraded service provisioning.

Keywords: Adaptive information system, reflective architecture

1 Introduction

In the last years, the design and development of information systems have significantly changed due to new network architectures and devices, which increase the number of distribution channels available for delivering of information or services. In the anytime/anywhere/anyone paradigm [18], a novel generation of applications [9] modify themselves according to the change of context, or to specific application constraints; for example, adaptive hypermedia applications [5,1, 20] modify data presentation according to the specific client browser capability. The goal of this paper is to present an architecture of a reflective adaptive system and adaptation strategies at different levels. First, we consider the possibility of modifying the distribution channel by means of adaptive information systems based on reflective architectures. The principle of reflection [10] is mainly studied in the programming language community and it consists in the possibility of

M.E. Orlowska et al. (Eds.): ICSOC 2003, LNCS 2910, pp. 115–131, 2003.

system inspecting and adapting itself by using appropriate metadata. In [6,12], the use of reflective architectures has been proposed as a new design principle of middleware, but the adaptivity is in charge of applications only; other papers [16,7] have considered the use of non-reflective adaptive channels, but they consider only the network as channel. Clearly adaptive distribution channels may be also used with adaptive applications to create a novel generation of multi-channel adaptive information systems. The middleware architecture we present allows overcoming existing limitations of information systems by means of modification of controllable components of distribution channels, identified through their description, and according to the specific context and level of Quality of Service (QoS) requested by users. The distribution channel delivers *e*-Services in order to satisfy a given QoS. If the user-specified quality level cannot be satisfied then our strategies try to adapt the distribution channel to a reduced QoS level still acceptable for the user. If the downgrading of QoS levels is still not sufficient, then other alternatives to provide the service are considered according to the users and service constraints.

The paper is organized as follows: in Section 2, we present the requirements of a reference example used to show, throughout the paper, how our architecture is able to adapt the distribution channel to deliver *e*-Services; Section 3 shows the distribution channel model representing metadata used by the reflective architecture. In Section 4, we present the general architecture and then we describe adaptive strategies in Section 5. Section 6 explains the adaptations functions used in our strategies. Finally, Section 7 discusses the state of art of adaptive information systems.

2 Reference Example

The reference example taken from the banking information system domain is based on a service that "*allow users, through Internet, to see in real time interviews with financial analysts*". This activity requires the delivery of real time videos along Internet, with some mandatory minimum requirements: the user device must be audio enabled and due to the real time nature of the service the channel bandwidth is also relevant. The information system has to negotiate with the user a reasonable minimum channel bandwidth that s/he accepts to see the video in an acceptable mode from her/his point of view.

3 Distribution Channel Model

The distribution channel model adopted in this paper defines the metadata needed for adaptation functionalities. Developing our first proposal in [13], we specify this information at three different levels of abstraction. The conceptual level specifies the general characteristics of service distribution; at logical level, we characterize such general characteristics; at technological level, technical tuning parameters are specified.

At each level, characteristics may be either observable or controllable, or both, yielding different levels of possible adaptation. In the following, we illustrate the three levels in detail.

3.1 Conceptual Model

We consider a conceptual distribution channel as an access point to a specific set of information or execution services. This level corresponds to channel definitions as they are viewed from the commercial point of view. For instance, a service can be provided via web, via a call center, or using short messages. In the description of distribution channels at conceptual level, a service can be viewed as a functionality with a set of constraints. If requirements for invoking the service on each channel on which it is provided, and for providing the results of the service itself, are given, the service is well described for the distribution channel.

At this level a distribution channel is defined as:

- **Set of services**: each distribution channel supplies a specific set of services
- **Technological features**: each distribution channel has specific technological features characterizing the definition of channel.

Reference example. Within a banking information system, we consider the Internet banking channel as the one requiring that customers interact with the Bank information system by means of an Internet connection. Customers access the Bank application (typically a web site) through a login/password mechanism. After the authentication phase, they may carry out services according to their personal profile. The technological constraints in this case require that the provider provides the service through a web server, a http connection, and a video-player plugged in the browser on the client side.

3.2 Logical Model

From the logical point of view, we characterize a distribution channel as composed of (see Fig.1):

- The *user device* of application users,
- The *network interface* through which the device is connected to the *network*,
- The *network* used to transfer information,
- The *application protocols* used by *e*-Services.

Each element composing the distribution channel is characterized by a number of *attributes* (e.g., device screen resolution or network topology). Each attribute is associated with a value, which can be numeric, as for example in the case of the device weight, or a set of numbers, or a mathematical function, e.g., the graph function describing the network topology. The service delivery is affected by user requirements. To specify them, we introduce the concept of *rating*

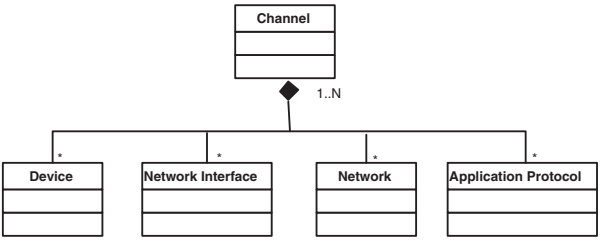

Fig. 1. UML specification of logical distribution channel components

class, which associates qualitative values (e.g., "fast" or "slow") to attributes in a given application domain. Rating classes are defined thanks to a measurement scale applied on measurable attributes. The scale has to be at least ordinal, but it can be also interval, ratio or absolute [15]. The rating classes are used to describe QoS levels offered by a given *e*-Service. An example of QoS levels is shown in Fig.2, where the quality dimension "speed" is associated with three different quality level: "very high", "high", and "medium". For each quality level a minimum value is defined.

```
<QualityLevels ServiceID="S1">
  <Dimension name="speed">
    <Level name="very high">
      <LogicalAttribute name="Bandwidth">
        <Condition type="greaterThan" unit="Kbps">512</Condition>
      </LogicalAttribute>
    </Level>
  </Dimension>
  <Dimension name="speed">
    <Level name="high">
      <LogicalAttribute name="Bandwidth">
        <Condition type="greaterThan" unit="Kbps">150</Condition>
      </LogicalAttribute>
    </Level>
  </Dimension>
  <Dimension name="speed">
    <Level name="medium">
      <LogicalAttribute name="Bandwidth">
        <Condition type="greaterThan" unit="Kbps">128</Condition>
      </LogicalAttribute>
    </Level>
  </Dimension>
  ...
</QualityLevels>
```

Fig. 2. Example of QoS levels

Reference example. According to the business requirements of the banking information system, we define the domain specific relevant attributes for all the four components previously defined.

Table 1. The 5 tuples of Internet banking channel used in the examples

Internet Banking	Device					Interface		Network			Protocol		
	Type	Screen resolution	Number of colors	Audio	Input Device	Type	Transfer rate	Type	Transfer rate	Security	Type	Standardisation	Security
n1	PC	C	C	O	O	Network Card	C	Wired	O	O	HTTP/SSL	O	O
n2	PC	C	C	O	O	Modem	C	Wired	O	O	HTTP/SSL	O	O
n3	TV	O	O	O	O	Modem	C	Wired	C	O	HTTP/SSL	O	O
n4	Mobile Phone	O	O	O	O	GPRS	C	GPRS	O	O	HTTP	O	O
n5	PDA	C	C	O	O	GPRS	C	GPRS	O	O	HTTP	O	O

C = Controllable attributes O = Observable attributes

The first element describing a distribution channel is the device on which the end-user interacts. Within the financial application domain, we consider as relevant attributes for this component: the *screen resolution* and the *number of colors*, which are relevant when the information system wants to send users graphical information. Other key attributes are the *audio support* describing the presence or absence of audio cards inside the device and the *input device* used by customers; this attribute is relevant in the definition of the best interaction methods. The second component is the network interface representing the connection between devices and transmission media. It is worth noting that a device can access different networks by means of different interfaces; for example, a PC can access the Internet via LAN through a network card or via PSTN by means of an analogical modem. In the financial context, we consider that the only relevant attribute is the *Transfer rate* achievable by the specific interface.

The next component describing a distribution channel is the network. It includes all physical structures, hardware components and protocols defining a network infrastructure. In this component, we include all protocols covering the first four levels of the ISO/OSI protocol stack. Within the bank information system, we identify as relevant attributes for the network the *transfer rate* and the *security* level that it offers.

We also consider as a distribution channel component the set of application protocols allowing users to interact with the information system. We identify two interesting attributes: the *security* support and the *standardization* of the application protocol, which is an important feature for reusing existing application parts.

3.3 Technological Model

From a technological point of view a distribution channel is defined by a tuple of specific instances of device, network interface, networks and application protocols. If a value does not exist we assume that it is *null*. At technological level, it

is possible to define attributes as *observable* (e.g. device position), if a software layer allows only showing their values to the information system; or *controllable* (e.g. bandwidth or screen resolution), if it is also possible to modify them. It is worth noting that a logical model can be instantiated with several tuples having at least one common attribute value. For example the logical distribution channel Internet (described in Section 3.2) is composed of a set of tuples each one characterized by the use of the HTTP as application protocol.

Reference example. An example of the technological model applied to Internet banking channel is shown in Table.1, where for each attribute and for each tuple, we indicate if it is observable (O) or controllable (C).

4 General Architecture

Fig.3 shows the general architecture and its relationships with clients and services. Three are the layers composing our architecture; they are:

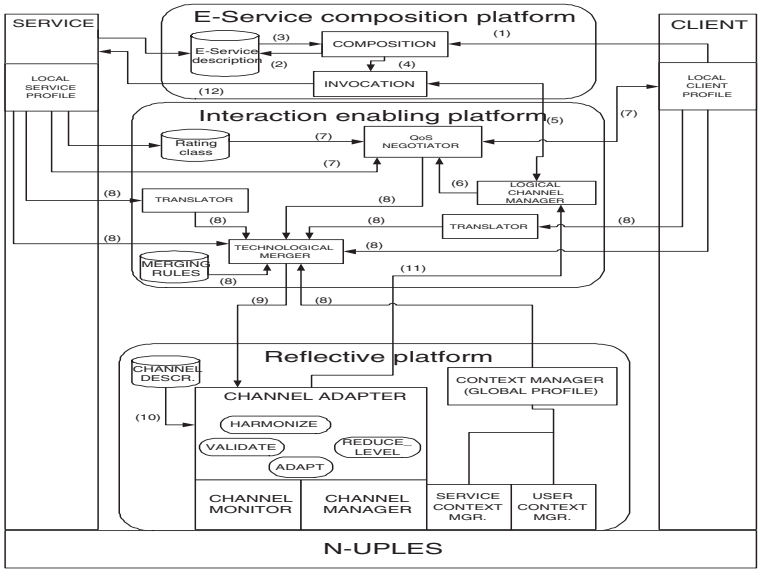

Fig. 3. General architecture

- *e-Service composition platform*, which is in charge of receiving the client request, selecting *e*-Service(s) satisfying it, and invoking the selected *e*-Service(s).
- *Interaction enabling platform*, which is the core of our architecture, because it is in charge of collecting constraints from *e*-Services, clients and context,

determining the QoS for each *e*-Service according to the client profile and selecting the best channel where the *e*-Service can be delivered.
- *Reflective platform*, which is in charge of adapting the selected distribution channel according to the constraints obtained from the Interaction enabling platform and monitoring if the distribution channel along which an *e*-Service is delivering respects the QoS level chosen by user.

Our architecture interacts with the client, which can be a user or a software agent, and with the *e*-Service. It is worth noting that the general architecture is decentralized so it is possible that all components of each layer are distributed over a number of hosts.

4.1 Constraints

In this paragraph we introduce the concept of technological constraints. They are given by service and user, through local and global profiles, and by context, and they are collected by *Technological merger* (see Fig.3), which sends them, opportunely integrated, to the Reflective platform.

Let T_c be the set of *Technological Constraints* associated with a tuple.

Because there exist different QoS levels that user might accept, there will exist a set of T_c for each tuple. We indicate with Δ_T this set of T_c.

Each element $T_{c_i}^j$ is defined as

$$T_{c_i}^j = \langle v_{min_i}, v_{max_i}, attribute, component \rangle$$

where:

- the index i represents the tuple and the index j the instance of Δ_T we are considering;
- v_{min} and v_{max} are the minimum and the maximum of a mathematic function (i.e. the media or the peak or simply the identity function) calculated for each distribution channel attribute value;
- *attribute* and *component* are respectively the attribute and the component where the constraint is defined.

4.2 E-service Composition Platform

The first layer of our architecture is in charge of receiving the client requests, defining the appropriate (set of) *e*-Service(s) satisfying them. The goal is obtained by analyzing the static description of the requested *e*-Services and the functional and non-functional requirements of the client as described in [3]. The *e*-Service composition platform selects the best *e*-Service, and requests the Interaction enabling platform to find the best distribution channel. When the distribution channel is selected, the *e*-Service composition platform invokes the execution of the *e*-Service.

Reference example. Let us assume that a user requests the video streaming service about an interview of financial analysts. He sends his request (through arc (1) of Fig.3) to the *Composition* module, which queries the *e*-Service description repository and gets the description of *e*-Service S1 only (arcs (2) and (3) of Fig.3). The *Composition* module calls the *Invocation* module which passes to the *Logical channel manager*, in the Interaction Enabling Platform, the request of enabling the delivery of *e*-Service S1 (arc (5)).

4.3 Interaction Enabling Platform

The Interaction enabling platform collects all requirements from the client, the *e*-Service and the distribution channel in order to manage service delivery on a given distribution channel. The *Logical channel manager* module, according to the description of the *e*-Service received from the *Invocation* module, selects a technological distribution channel (that is, a given tuple). The *QoS negotiator*, invoked by the *Logical channel manager* (arc (6)), requests the sub-set of the local client profile involved in the definition of QoS levels related to the given *e*-Service. This information can be stored directly in the user device or in other hosts if the device has a very small amount of memory. The QoS levels acceptable for the end users are chosen among the ones available for the *e*-Service stored in the Rating class repository of Fig.3 (arc (7)). These different levels allow a flexible QoS managing policy, by modeling different satisfaction degrees for the client. The *Technological merger* module receives inputs (arc (8) of Fig.3) from the *QoS negotiator*, and non negotiable constraints from both client and service profiles. We identify two kinds of constraints: logical and technological. The former are related to logical features which do not refer to measurable attributes and they have to be translated into constraints on technological attributes. An example of logical attribute is the device graphical capability, which is the result of both screen resolution and number of colors of a given device at technological level. Technological constraints are related to technological attributes as described in Section 4.1. Logical constraints, coming from both user and service profile, are first translated into technological ones (by using appropriate *Translator* modules, which are different for both client and service). The *Technological merger* integrates all constraints by using the Merging rules repository and passes down to the Reflective platform the result of its elaboration and QoS levels expressed as sets of technological constraints, requesting to satisfy them.

Reference example. Continuing the example, the *Logical channel manager* invokes the *QoS negotiator* by passing the description of *e*-Service S1. This component finds in the Rating class repository the QoS levels shown in Fig.2.

In the example of Fig.2 the *e*-Service S1 is offered at several different QoS levels about the channel speed ("Very high", "high", "medium", etc.). The client (through his local profile) selects his best and worst acceptable level by defining a sorted sub set of QoS. They are then translated into technological constraints before passing them to the *Technological merger*. The translation considers that

channel bandwidth depends on both the transfer rate of the Network interface and the Network. The *Technological merger* receives also another technological constraint specified by the *e*-Service S1 requiring that the client device must have the audio card turned on to allow user to listen to the interview. This requirement is mandatory in order to deliver the *e*-Service; consequently it has not been included in the definition of the QoS level.

Let $n3$, described in Table 1, be the tuple chosen by the *Logical channel manager*. Let be

$$\begin{cases} \langle 0, 128kb/s \rangle \text{ for Transfer rate of Interface} \\ \langle 0, 10Mb/s \rangle \text{ for Transfer rate of Network} \\ \langle Off, On \rangle \text{ for Audio of Device} \end{cases}$$

the range of values in which the distribution channel we are considering can work.

The *Technological merger* module generates the following constraints:

$$\begin{cases} \text{level 0} \begin{cases} \langle 150kb/s, *, Transfer\ rate, Interface \rangle \\ \langle 150kb/s, *, Transfer\ rate, Network \rangle \\ \langle On, On, Audio, Device \rangle \end{cases} \\ \text{level 1} \begin{cases} \langle 128kb/s, *, Transfer\ rate, Interface \rangle \\ \langle 128kb/s, *, Transfer\ rate, Network \rangle \\ \langle On, On, Audio, Device \rangle \end{cases} \end{cases}$$

where $*$ means that a maximum value is not given.

This output represents the two acceptable technological QoS levels for the user. The chosen tuple and the set of technological QoS levels, that is the set of technological constraints, are sent to the Reflective platform.

4.4 Reflective Platform

The central element of the Reflective platform is constituted by metadata, that is, the description of distribution channels built by using the model shown in Section 3 and stored in the Channel description repository. The availability of this characterization allows the platform to evaluate if constraints can be satisfied.

The *Channel monitor* module is in charge of measuring the current values of the attributes describing the selected distribution channel. Its information is used by the *Channel adapter* module, which first evaluates if all constraints can be satisfied for the tuple chosen by the upper layer from a hypothetical point of view; that happens if the current working point of tuple satisfies constraints or if it is possible to modify the distribution channel attributes values according to constraints. If there is the theoretical possibility of adaptation, then the *Channel manager* module realizes the modification of the distribution channel by invoking the appropriate software components. If one or more attributes cannot be modified, for example because the network manager does not accept the request of additional bandwidth, the Reflective platform reduces QoS levels and tries to modify the current tuple according to the new values. The reduction of QoS levels is executed also if there are no acceptable hypothetical solutions. If

the Reflective platform does not satisfy the client request at least at the lower level of the QoS assigned, then it advises the *Logical channel manager* module (arc 11 in Fig.3), which will select another tuple according to the *e*-Service description. The Reflective platform has also the goal of monitoring and, if necessary, adapting tuples along which *e*-Services are delivering services. Finally the *Service context manager* and the *User context manager* modules measure the behavior of *e*-Services and users to build their global profiles (realized by the *Context manager* module).

5 Adaptive Strategies

We consider that the adaptation of distribution channels to services means to find and, in case, modify, if it exits, a tuple satisfying all constraints.

Let a working point (Wp) be the set of all attribute values carried out by Channel Monitor in a given instant for the given tuple.

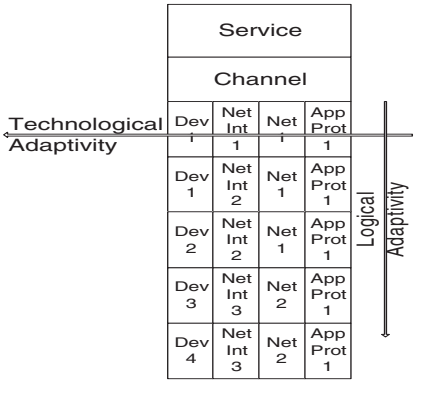

Fig. 4. Channel adaptivity

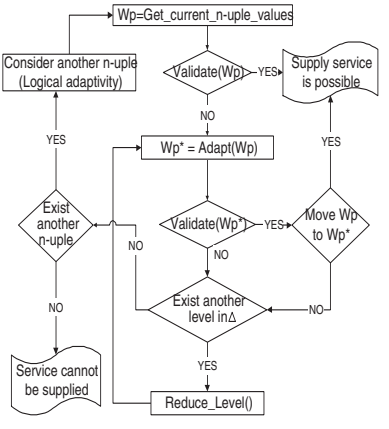

Fig. 5. Adaptive flow-chart

Fig.4 shows the two strategies of adaptivity we consider; logical and technological, which are developed in sequence, in particular:

- The "*Technological Adaptivity*" represents the tuning phase on the single tuple, i.e. the attempt to change some attributes of the given tuple to satisfy constraints, and it is realized by the Reflective platform
- The "*Logical Adaptivity*" represents the possibility of using a different tuple of the same logical distribution channel, it is realized by the Interactive enabling platform.

Fig.5 shows the flow-chart of adaptive strategies mainly realized by the *Channel adapter* module, while the concrete movement of working point is realized by the

Channel manager and the selection of alternative tuples is in charge of the Interaction enabling platform. Hereafter we consider the "Technological Adaptivity" only.

The $Validate()$ function evaluates if the current tuple working point satisfies the conditions required by constraints. In this case the service is supplied along the current tuple, otherwise the *Adapt*() function (see Section 6.2) proposes an acceptable working point available for such tuple (Wp^*). The new point is, then, evaluated again by the $Validate()$ function. If it satisfies constraints then the *Channel manager* module tries to change the current working point of the selected tuple. If it is impossible to move the current tuple working point, due to technical limitations or because Wp^* does not satisfy the constraints, then the *Channel adapter*, by means of the *Reduce_level*() function, tries to progressively reduce QoS levels, according to the service and user requests, looking for an existing QoS level satisfying them. After each invocation of the *Reduce_level*() function, the *Adapt*() function is invoked.

If technological adaptation fails, the information system, within the Interaction enabling platform, selects another tuple, QoS levels and constraints, and the technological adaptive strategy starts again.

This iterative process ends when a tuple satisfying all constraints is found or when no tuple can deliver the service. In this case the *Logical channel manager* tries to use multichannel delivery strategies selecting an alternative delivery channel, if possible, for the given *e*-Service.

6 Adaptive Functions

As shown before, adaptive strategies are executed in two phases: theoretical and practical. For the theoretical study the *Channel adapter* module has to know the channel structure. It obtains this information from the Channel description repository. It is worth noting that the designer has to write some functions defining the relationship between two or more technological constraints of a specific tuple to express them in a compatible way. These functions are named *Harmonize Functions* as shown in Fig.3.

Following the direction of most recent literature [2], to adapt means looking for an admissible solution minimizing the distance between service directive and channel availability to maximizing the QoS. It is clear that this concept of QoS is dynamic [17,14,19], because channel conditions vary both in time and space.

The *Adapt*() function realizes such algorithm, while the $Validate()$ function detects if the working point, current or proposed by the *Adapt*() function, satisfies the constraints. In the next sections we explain more in depth these functions.

6.1 Validate Function

The $Validate()$ function evaluates if current attribute values, that is, the current working point, (Wp), satisfy all constraints. Formally

$$Validate() : \{A\} \rightarrow \{[0,1] \in N\}$$

where $A = \langle A_o \cup A_c \rangle$ is the set of attributes composed by the union of observable (A_o) and controllable attributes (A_c) of a given tuple. The result of $Validate$ function is 1 if Wp satisfies all constraints, it is 0 otherwise.

Let C_A the hypercube formed by the union of theoretical available working points included between all v_{min} and all v_{max} carried out from Interaction enabling platform and let C_C the hypercube formed by all theoretical working points of a given tuple; it results that acceptable solutions are those included in the intersection of C_A and C_C.

Each edge hypercube C_{C_i} (resp. C_{A_i}) of C_C (resp. C_A) associated with the generic attribute (i) is defined as follows:

$$\begin{cases} [v_{i_{min}}; \infty[& \text{if constraints are like } \langle v_{i_{min}}, *, i, component \rangle \\ [v_{i_{min}}; v_{i_{max}}] & \text{if constraints are like } \langle v_{i_{min}}, v_{i_{max}}, i, component \rangle \\ [0; v_{i_{max}}] & \text{if constraints are like } \langle *, v_{i_{max}}, i, component \rangle \\ [0; \infty] & \text{if there are no associated constraints} \end{cases}$$

It is worth noting that an attribute domain can be discrete rather than continuous and all Wp are included in C_C.

The $Validate()$ function is defined as follows:

$$Validate(Wp) \mid_{C_A, C_C} = \begin{cases} 1 & \text{if } Wp \in (C_C \cap C_A) \\ 0 & \text{otherwise} \end{cases}$$

A hypercube can have an infinite extension along variables associated with constraints where $v_{i_{max}}$ is not defined. Moreover if an attribute allows only a value, the hypercube loses one dimension.

According to the flow-chart of Fig.5 the $Validate()$ function is also invoked each time a new Wp is proposed by the $Adapt()$ function as shown in next Section.

6.2 Adapt Function

The $Adapt()$ function is used to move the current channel working point to C_A hypercube, that is, to a new one respecting the constraints requested from e-Service and user. Thus it receives as input a working point (Wp) and return as output another working point (Wp^*).

Notice that $Adapt()$ function does not change the working point, it only detects if there exists a hypothetical point in the solution space where all constraints are satisfied. The result of function is:

$$Wp^* = Adapt(Wp) \mid_{(C_A, C_C)} = argmin(\| Wp; C_A \|)$$

The $Adapt()$ function looks for a point in C_C having the minimum distance from C_A.

Fig.6 shows a graphical representation of the $Adapt()$ function by considering only two continuous attributes. There exists an overlapping area between the two

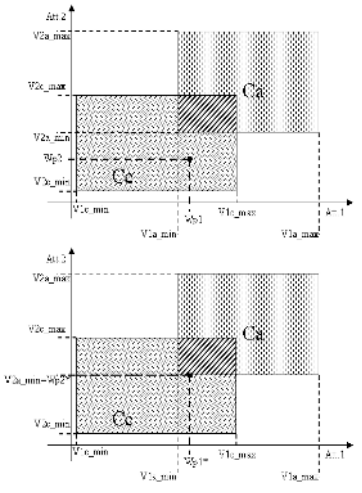

Fig. 6. Adapt() function behavior when there exists a no-empty intersection

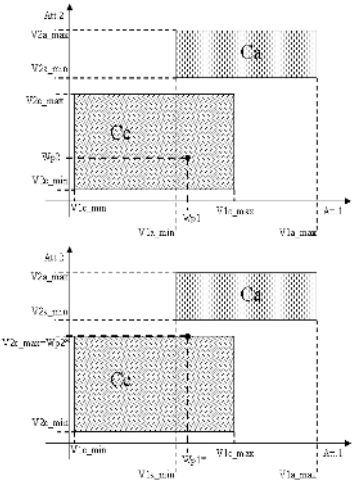

Fig. 7. Adapt() function behavior when there exists an empty intersection

areas C_A and C_C; consequently the *Adapt*() function will define a Wp^* as shown. Conversely, Fig.7 shows the case in which there exists an empty intersection between the two constraint sets. In this case the *Adapt*() function returns a new Wp^* point, that is the best working point available for the current tuple, however, using $Validate()$ functions the *Channel adapter* module will notice that Wp^* does not satisfy constraints and thus it will try to relax one or more constraints (that is to widen C_A) by using the *Reduce_Level*() function described in Section 6.3.

Reference example. Considering that the current tuples is $n3$, let C_A be:

$$C_A = \begin{cases} \langle 150kb/s, \infty \rangle & \text{for Network} \\ \langle 150kb/s, \infty \rangle & \text{for Interface Network} \\ \langle On, On \rangle & \text{for Audio} \end{cases}$$

where values are related to the best QoS level chosen by user.

Let C_C be the following:

$$C_{C_{n3}} = \begin{cases} \langle 0kb/s, 128kb/s \rangle & \text{for Network} \\ \langle 0kb/s, 128kb/s \rangle & \text{for Interface Network} \\ \langle Off, On \rangle & \text{for Audio} \end{cases}$$

Let we suppose that the current working point is:

$$Wp = \begin{cases} 64\ kb/s \text{ for Network} \\ 64\ kb/s \text{ for Interface Network} \\ On \quad \text{for Audio} \end{cases}$$

Calculating the $Validate()$ function on these values we obtain

$$Validate(Wp)\ |_{C_A,C_C} = 0$$

Then the *Channel adapter* module tries to adapt distribution channel by calculating the $Adapt()$ function:

$$Wp^* = Adapt(Wp) = \begin{cases} 128\ kb/s \text{ for Network} \\ 128\ kb/s \text{ for Interface Network} \\ On \quad \text{for Audio} \end{cases}$$

but the output of $Validate()$ is still the same

$$Validate(Wp^*)\ |_{C_A,C_C} = 0$$

6.3 Reduce_level Function

Let Δ_T be the ordered set of QoS levels acceptable for a given tuple by service and user, generated by the Interaction enabling platform. Each QoS level is composed by a set of constraints T_c (Section 4.1), that is different levels may have different T_c. The $Reduce\_level()$ function tries to downgrade the C_A of one level. This operation is strongly related to the tuple the *Channel adapter* module is considering; so the function tries to relax attribute constraints that are still not respected looking for closest downgrading level allowed for the tuple. Formally:

$$Reduce\_level() : \{C_A\} \rightarrow \{C_A\}$$

In particular

$$C_A^{new} = Reduce\_level(C_A)\ |_{\Delta_T}$$

It is hopefully that this new C_A is formed by satisfable constraints for the channel characteristic. The reduction process is progressive and it ends if no other level is available or if an acceptable one exists.

Reference example. By continuing the reference example the *Channel adapter* module controls if it is possible to provide a downgraded service. For this goal it uses the $Reduce\_level()$ function.

$$C_A^{new} = Reduce\_level(C_A)\ |_{\Delta_T} = \begin{cases} \langle 128Kb/s, \infty \rangle \\ \langle 128Kb/s, \infty \rangle \\ \langle On, On \rangle \end{cases}$$

Consequently, by remembering that

$$Wp^* = \begin{cases} 128\ kb/s & \text{for Network} \\ 128\ kb/s & \text{for Interface Network} \\ On & \text{for Audio} \end{cases}$$

we obtain $$Validate(Wp^*)\ |_{C_A,C_C} = 1$$

because $$(C_C \cap C_A \neq \emptyset)\ \wedge\ (Wp^* \in C_C \cap C_A)$$

That is, the current working point is now acceptable and it is the same of the lower bound asked after relaxation. We suppose that all the modification of attribute values are executed without failures; the *e*-Service provisioning can now start (arc (12) in Fig.3) and it will be monitored by *Channel monitor* module in order to respect the requested constraints during all the provisioning period.

7 Related Work

The field of adaptive information systems is new and relatively unexplored. The use of reflective architecture for mobile middleware is discussed in [12,4]; other work [6] use the reflection principle to support the dynamic adaptation of applications. However, none of these papers mention the possibility to assign the task of adaptation to the distribution channel instead of application; in our opinion, our approach simplifies the design and implementation of *e*-Services because they do not have to realize adaptive behaviors.

The concept of dynamic service provisioning and dynamic QoS [19,14] involves both channel and application issues; moreover the new wireless networks and devices increase the complexity of solutions. Different approaches are being developed; in [7,16] the possibility of adapting the network by maximizing the network efficiency looking at its intrinsic parameters is investigated. In the same direction other systems [2] try to consider features regarding users, but their channel idea is simply the network and their adaptation process consider a simple user-profile containing information about user preferences and past behavior. Other approaches [1,20] try to adapt applications to the distribution channel; they consider the channel as an only-observable system and try to adapt the application to it; some systems put their attention above all on the presentation of the information delivering on a channel; important examples of this approach are defined in the field of adaptive hypermedia [8,5].

The enriched definition of distribution channel [13] distinguishes our approach from others, because we consider a distribution channel as a tuple of four different components and the network is only one of them, thus our approach is at a higher level of abstraction. Moreover our strategies try to adapt the distribution channel before trying adaptive application strategy to supply at least a downgraded service.

8 Conclusion

In this paper we presented adaptive information systems to deliver *e*-Services by means of reflective architectures. The use of reflection principle, obtained by using a high level description of distribution channels, allow us to define adaptive strategies to modify the distribution channel itself (viewed as a tuple of four different components) to *e*-Services. Two levels of adaptability are considered: logical and technological. The former, realized in the Interaction enabling platform, selects the tuple where the *e*-Service has to be delivered. The latter strategy tries to modify the controllable attributes of current tuple to satisfy constraints. If it is impossible to adapt the selected tuple, then the Interaction enabling platform selects an alternative tuple. This is the first proposal to design a new generation of multichannel adaptive information systems. We are now studying more efficient logical adaptive strategies in the selection of the tuple where to deliver the service, and how to assign each architectural components in a distributed information system; we are also evaluating how a service can select another logical distribution channel. Another problem we are studying is channel monitoring in mobile systems and adaptive networks.

Acknowledgments. This work has been developed within the Italian MURST-FIRB Project MAIS (Multi-channel Adaptive Information Systems) [11].

References

1. G. Ammendola, A. Andreadis, and G. Giambene, *A software architecture for the provision of mobile information services*, Softcom, International Conference on Software, Telecommunications and Computer Networks (Dubrovnik (Croatia) and Ancona, Venice (Italy)), October 2002.
2. G. Araniti, P. De Meo, A. Iera, and D. Ursino, *Adaptively control the QoS of multimedia wireless applications through "user profiling" techniques*, IEEE Journal On Selected Areas in Communications (JSAC) (2003), Forthcoming.
3. L. Baresi, D. Bianchini, V. De Antonellis, M.G. Fugini, B. Pernici, and P. Plebani, *Context-aware composition of e-services*, In Proc. of VLDB Workshop on Technologies for E-Services (TES'03) (Berlin, Germany), 2003.
4. G. Blair, A. Andersen, L. Blair, G. Coulson, and D. Gancedo, *Supporting dynamic QoS management functions in a reflective middleware platform*, IEEE Proceedings – Software, 2000.
5. P. Brusilovky, *Adaptive hypermedia*, User Modeling and User Adapted Interaction **11** (2001), no. 1–2, 87–100.
6. L. Capra, W. Emmerich, and C. Mascolo, *Reflective middleware solutions for context-aware applications*, Lecture Notes in Computer Science **2192** (2001).
7. A. Hac and A. Armstrong, *Resource allocation scheme for QoS provisioning in microcellular networks carrying multimedia traffic*, International Journal of Network Management **11** (2001), no. 5, 277–307.
8. A. Kobsa, J. Koenemann, and W. Pohl, *Personalized hypermedia presentation techniques for improving online customer relationships*, The Knowledge Engineering Review **16** (2001), no. 2, 111–155.

9. J. Krogstie, *Requirement engineering for mobile information systems*, Proc. of International Workshop on Requirements Engineering: Foundation for Software Quality (Interlaken, Switzerland), 2001.
10. P. Maes, *Concepts and experiments in computational reflection*, In Proc. of Object-Oriented Programming Systems, Languages, and Applications (OOPSLA) (Orlando, Florida, USA), vol. 7, ACM Press, 1987, pp. 147–155.
11. MAIS_Consortium, *MAIS: Multichannel Adaptive Information Systems*, http://black.elet.polimi.it/mais/.
12. V. Marangozova and F. Boyer, *Using reflective features to support mobile users*, In Walter Cazzola, Shigeru Chiba, and Thomas Ledoux, editors, On-Line Proceedings of ECOOP'2000 Workshop on Reflection and Metalevel Architectures, June, 2000.
13. A. Maurino, B. Pernici, and F.A. Schreiber, *Adaptive behaviour in financial information system*, Workshop on Ubiquitous Mobile Information and Collaboration Systems (Klagenfurt/Velden, Austria), June, 2003.
14. K. Nahrstedt, D. Xu, D. Wichadakul, and B. Li, *QoS-aware middleware for ubiquitous and heterogeneous environments*, IEEE Communications Magazine **39** (2001), no. (11), 140–148.
15. N.Fenton, *Software metrics, a rigorous approach*, Chapmann & Hall, 1991.
16. R. Raymond, F. Liao, and A. T. Campbell, *A utility-based approach for quantitative adaptation in wireless packet networks*, Wireless Networks **7** (2001), no. 5, 541–557.
17. D. Reiniger, R. Izmalov, B. Rajagopalan, M. Ott, and D. Raychaudhuri, *Soft Qos control in the watmnet broadband wireless system*, IEEE Personal Communications Magazine **Feb** (1999), 34–43.
18. K. Siau, E.P. Lim, and Z. Shen, *Mobile commerce: Promises, challenges, and research agenda*, Journal of Database Management **12** (2001).
19. R. Steinmetz and L. Wolf, *Quality of service: Where are we?*, Proc. of IFIP International Workshop on Quality of Service (IWQOS'97) (New York City, New York, USA), IEEE Press, 1997, pp. 211–222.
20. V. Zariskas, G. Papatzanis, and C. Stephanidis, *An architecture for a self-adapting information system for tourists*, Proc. of the Workshop on Multiple User Interfaces over the Internet: Engineering and Applications Trends (in conjunction with HCI-IHM'2001) (Lille, France), 2001.