A Rule-Based Language for the Specification of Message Routing Policies in a Universal Communication System

Elisa Bertino
Dip. di Scienze dell'Informazione
Università degli Studi di Milano
V.Comelico, 39/41 20135 Milano, Italy
bertino@dsi.unimi.it

Munir Cochinwala
Telcordia Technologies
(formerly Bellcore)
445, South St., Morristown, NJ.USA
munir@research.telcordia.com

Marco Mesiti
Dip. di Info. e Scienze dell'Informazione
Università degli Studi di Genova
V.Dodecaneso, 35 16146 Genova, Italy
mesiti@disi.unige.it

Abstract

In this paper we propose a declarative rule language, based on the ECA paradigm [7], for specifying message filtering and routing policies in a Universal Communication Identifier (UCI) system. A user subscribing to a UCI system has a unique identifier, independent from the actual communication devices the user owns. It is a task of the UCI system to properly route messages or phone calls to the proper user devices, according to the type of message, the type of device, and to the user preferences. The rule language presented in this paper allows users to state their personal policies for message/phone call routing and filtering. Those policies, expressed through the rule language, are incorporated into the user profile and used by the UCI system in order to perform message/phone call dispatching. In the paper, we first present the rule language, and discuss the rule evaluation and execution process. We then present an overview of an UCI system implementing the proposed language.

1 Introduction

Recent advances in technologies for communication devices, especially portable ones, and in networks have made available a large variety of means by which users can easily communicate among each others anywhere and at any time. Users have now several options among which to choose whenever they need to communicate and/or to exchange data, including multimedia ones, with other users or applications. Devices, such as new generation cellular phones, palm PC, PDA, laptops, have enormously improved the communication process, by increasing both the quantity and quality of data that are exchanged and by providing easy-to-use interfaces.

Such a wealth of communication devices has, however, the problem that now users end up having several phone numbers and other electronic addresses at which they can be reached. A user has then to let others know all the various numbers and addresses where he/she can be reached. Things become more complicated when a user can be reached at a certain number, or address, depending on some conditions, such as the time of day, or when a user needs to be selective when letting others know his numbers or addresses. Things are also complicated from the side of the caller. A user wishing to get in touch with another user has to know which number or address to use, possibly depending on some conditions, such as the time of day or the type and contents of communication to be made.

Approaches, which have been recently proposed to address such a problem, are based on the concept of *Universal Communication Identifier* (*UCI*) [2]. The UCI is a unique identifier associated with a person independently from the devices he/she has. A user wishing to send a message to a given user, or to call such a user, uses such identifier. The message or call is then automatically routed by the UCI system to the appropriate device number or electronic address. In this way the message sender/caller is free from keeping track of the devices currently belonging to the message/call receiver. Such an approach lets users free to concentrate on the communication contents rather than on details concerning the communication devices.

In order to be effective, such an approach must provide a mechanism by which UCI owners can specify policies for routing messages and calls arriving at their UCI. Such policies can depend on a variety of conditions, such as the sensitivity of the message contents, the sender/caller characteristics, the time at which the message/call arrives, the availability/capability of the devices. Moreover, the UCI owners must also be able to specify policies concerning messages/call arriving at a particular device, in order to define a particular behavior for such device. An example of such a policy is “reject messages containing huge images arriving at my laptop”. Finally, the UCI owner should be able to require notification messages whenever a message/call arrives to his/her UCI, or a particular device handles a message/call, or a message/call is rejected.

In this paper we address such a requirement by introducing a declarative *Policy Language* that supports the specification of message/call routing policies for the UCI owner. The language is based on the active rule paradigm, typical of Active Databases [7], and makes it possible for a user to specify his/her own routing policies through a set of *policy rules*. A policy rule specifies the *routing action* to be taken whenever a message/call, arriving at the UCI or at one the devices associated with it, verifies a given *condition*. Possible actions that can be taken include rejecting the message/call, routing the message/call to a device or a set of devices, sending notification messages. Conditions specify filters on the applicability of the rule

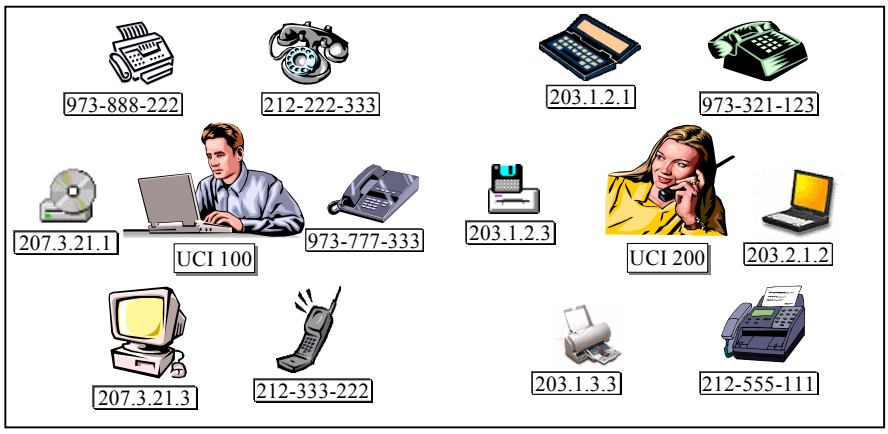

Figure 1: Two users, subscribing to the UCI service, and their devices

and can be expressed against the sender properties, the device capabilities and states, the message itself. Because the language is declarative, specifying new routing policies it is very easy. To further enhance extensibility, the language supports the notions of device type, message type, and user credential type, representing respectively the properties associated with device, messages, and users. Such properties are important since they are the basis on which filtering and routing conditions can be specified in our rule language. All those types are organized according to an inheritance hierarchy that can be refined by the introduction of new types, following an object-oriented approach.

The Policy Language has been implemented into a Policy Engine able to enforce the policies specified by the users subscribing to the UCI services. We refer to such users as *Policy Engine Users*. Moreover, by means of translation services that external providers can integrate into the Policy Engine, it is possible to translate a message format whenever a device is not able to handle the message directly. For example, an email can be translated into a speech if the device that should handle it is a phone. Other services can be coupled with the Policy Engine, such as services for checking the current state of a device (e.g. a mobile phone can be available, busy or unreachable), or for establishing the connection with the receiver's device. The Policy Engine is also equipped with a "parking queue" where messages that cannot be delivered because the receiver's devices are not available, can be temporarily stored. Finally to improve usability, two different end-user environments complement the Policy Engine, supporting respectively the system administrator and the policy engine users. The latter environment enables Policy Engine users to enter their preferences concerning message/call routing and filtering by using a form-based interface.

As far as we know, the policy language we propose is the first declarative language for expressing rules for routing and re-routing multimedia messages from a device to others and for specifying notification messages. However, various issues concerning message routing, filtering and notification have already been partially addressed by other systems, such as Local Number Portability [3], translation of 800 numbers [4], and Unified Message System [5]. The declarative language we propose smoothly combines the relevant features of all such systems and can be used directly by end-users for specifying the policy rules. By contrast in most of such systems, only administrators can specify such policies. A language based on a Prolog notation for routing email messages arriving at an email address to folders has been proposed in [6]. In such an approach a message arriving at an email address is stored in a particular folder based on filters defined by the message recipient. However, this approach only considers textual messages and does not support message re-routing from a folder to another. By contrast, our approach covers both multimedia messages and phone calls, by providing a language with a large variety of specialized conditions, and supports facilities for message/call re-routing and notification.

The paper is organized as follows. Next section introduces the relevant requirements for a language able to express policy rules. Section 3 introduces the object-oriented representation of the components of the communication process - we call them actors-, that is, devices, messages and users. Section 4 presents the policy rule language, whereas Section 5 briefly discusses the more relevant features of the Policy Engine architecture. Finally, Section 6 concludes the paper and outlines future research directions. Appendix A presents the grammar of the policy rule language, whereas some examples of message delivery are shown in Appendix B. For sake of simplicity, in the remainder, we will use the term 'message' to denote an actual message or a phone call, whenever no distinction is necessary.

2 Requirements for a Policy Rule Language

In this section we briefly discuss the relevant requirements for a language able to express policy rules. An important aspect of our approach is that it is based on filtering conditions establishing which rules apply to which messages. Therefore, the condition component of the language should be expressive and able to cover a variety

of situations. Moreover, the language should allow one to identify the devices that will handle a given message or receive notifications. Such devices should be identified based on their capabilities and states, and on the preferences specified by the owner. Moreover, the language should give the possibility of requiring a message re-routing action whenever a device is not available.

In the remainder of the section we first outline the main requirements concerning the filtering conditions, we then describe how the devices where a message should be routed or to be notified can be specified.

In the discussion we refer to the running example illustrated in Figure 1. Bob, in the left hand side of the figure, has the UCI 100 and Alice, in the right hand side of the figure, has the UCI 200. Bob has a phone at home, a phone at the office, a mobile phone, a PC, a DVD reader, and a fax machine; whereas Alice has a phone, a laptop, a fax machine, a printer, a data storage device, and a palm top. Each such device has an identification number, also shown in Figure 1.

2.1 Conditions for message filtering and routing

In order to devise possible requirements against a language for expressing filtering and routing conditions, we have identified the main "actors" of the message routing process. These are: the message itself, the sender of the message, the device on which the message is to be routed. In addition to the explicit actors of the communication process, some important conditions may concern environmental and contextual information. In this respect, we have identified as a relevant category, conditions concerning temporal information. We have thus classified possible conditions into four different categories, namely conditions on the message itself, on the sender characteristics, on the device, and temporal conditions, that we briefly discuss in what follows.

- *Conditions on the message.* These include conditions on the message type (e.g. audio, stream, email, voice), on the level of sensitivity of the information contained in the message (e.g. private vs. public information), on the encryption and compression status, and on the kind of device required for handling the message. In the running example, Bob can specify conditions such as: "if the message contains a video", "if the message arrives from a mobile phone", "if the message is encrypted", "if the message is compressed".
- *Conditions on sender characteristics*. These include conditions on sender name, age, nationality, company for which he/she works, and so on. In the running example, Bob can specify conditions such as: "if the sender is Alice", "if the sender age is less than 18", "if the sender is anonymous". Moreover, Bob can create groups of users and define filters such as "if sender belongs to group `FRIENDS`" or "if sender does not belong to group `FRIENDS`".
- *Conditions on device capabilities, state, security levels and ownership*. These include conditions on: the device status (e.g. available, busy, unreachable); the device capabilities, such as "if the device supports video stream", "if the device has a display"; the security levels of the device, such as "the device can be reached only by the owner or it is in a common room accessible to everyone"; the device ownership, such as "the device has been borrowed by another user".
- *Temporal conditions*. These include conditions on the time interval during which the message is delivered (e.g. between 5 p.m. and 7 p.m., or between 9/10/2000 and 9/15/2000), on the day of the week (or on the month of the year) of the message arrival (e.g. on Sunday, or, on August), on aggregations of days (e.g. on weekend, or on working week), for example, "if the message arrives on a weekday", or "if the message arrives during the month of May".

A suitable language should thus cover all above conditions. Moreover, it should support the specification of composite conditions, consisting of boolean combinations of simple conditions, so that users can specify complex filtering conditions.

2.2 Routing and Notification Services

The language should give the possibility of specifying the devices that have to handle a message by listing their device identifiers or expressing a condition on their capabilities, states or security levels. This specification should be possible either at UCI level or at single device level. The specification at the UCI level is required when a message arrives at a UCI and the system needs to select the devices for handling it. By contrast, the specification at single device level is required when a message, already routed to a device, cannot be handled by this device and, thus, other devices must be identified for such purpose. In both cases, the language should give the possibility of requiring translation of the message contents into another format and/or the generation of notification messages. Translating a message content into another format (for example translating a voice message into an email) is a crucial option in order to deal with the lack of specific devices. The UCI owner should specify the translation option required by using some special-purpose clauses of the rule language. Alternatively, the translation is performed automatically whenever the selected device is not able to handle the message and a translation service is available for translating the message from its format to one of those supported by the device.

Another important requirement is to provide some form of notification services. A *notification message* is a message containing the information about the delivery status of the message. Thus, a notification message can be issued for informing the UCI owner that a message arrived at the UCI, or that a device handled it, or that the message has been re-routed to another device, or rejected. For example, Bob may wish to receive a notification on the mobile phone whenever the fax machine is printing a message. By contrast, Alice may wish to reject all messages

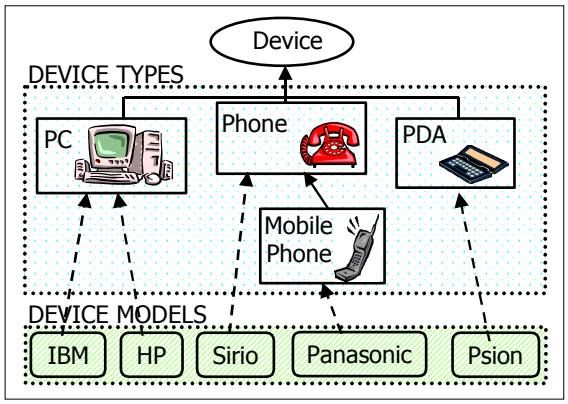

Figure 2: Device type hierarchy and device models

from Bob but be notified when he sends her a message. Finally, Alice may wish to be notified when a message cannot be delivered because she does not have a device able to handle the message.

3 Object-Oriented Representation of main Actors of the Communication Process

In this section we outline the most relevant characteristics of the main actors of the UCI system, namely the devices, the users and the messages that are described respectively in Subsections 3.1, 3.2 and 3.3.

3.1 Representation of Devices

Each device is characterized by a set of properties representing its capabilities, the states in which it can be found (e.g., available, busy, unreachable), the supported message formats (image, audio, SMS, fax and so on), and the security level. To provide an accurate representation of device capabilities at different levels of details, we have used in our model the concepts of *device type*, *device model* and single *device*. Our representation for devices can be mapped into the CC/PP (composite capabilities/ preference profiles) profile proposed by W3C [9]. We do not support such feature yet, because the CC/PP is still a working draft.

Device Type. A device type represents a group of devices with common capabilities independently from the company producing the specific devices. Examples of device types are: phone, mobile phone, PC, palm top. Each device type has a set of capabilities. Examples of capabilities are `displaySize`, representing the size of the display, `speaker`, representing the type of speaker and the watt power supported, `webCam`, representing the number of colors supported, the number and type of lens, the focus range and so on.

The various device types are organized according to an inheritance hierarchy. The root of such hierarchy is a device type representing the capabilities common to all device types. The hierarchy is extensible so that new device types can be introduced. The top part of Figure 2 shows some of the device types currently included in our system. Note from the figure that `PC`, `Phone` and `PDA` are defined directly from the root, whereas `Mobile Phone` is defined as a subtype of `Phone`, because a mobile phone is a specialized type of phone.

Device Model. Device models represent groups of devices of the same type, but produced by different companies. For example, `Panasonic GD90` is a model of `Mobile Phone`. Thus, a device model assumes the same capabilities of the device type with additional capabilities typical of such model. In the bottom side of Figure 2 we report some examples of device models associated with the corresponding device type.

An important aspect in modeling devices is represented by the device states. Indeed, several meaningful conditions for message filtering and routing are based on device states. An example is the state 'busy' for a phone. For each device model we thus maintain the possible states in which devices of this model can be. The states we consider are those that can be detected through services and functions provided by the network management system.

Device. A *device* is an instance of a specific device model associated with a user. Each device registered with the UCI system is characterized by a set of properties that we report in Figure 3. Users and system administrators can query those properties and conditions can be posed against them as part of the routing rules. Note, among other things, that each device has a unique number identifying it. For example, given the device model `Panasonic GD90`, Bob has a device of such model identified by number `212-333-222`, the corresponding phone number. To simplify device identification, however, a nickname can be associated with a device. The user can use it whenever the device identifier is expected.

A novel feature of our model is that each device has a security level. The security level is a value in the set {`HIGH`, `NORMAL`, `LOW`} specifying if the device is very secure (security level equal to `HIGH`) because the owner has it always with him/her (e.g. a mobile phone the owner always carries with him/her), not secure at all (security level equal to `LOW`) because the device can be accessed by everyone (e.g. a FAX machine of a department), or normally secure (security level equal to `NORMAL`), because normally accessed by the owner and trusted people.

A device can be in one of the states specified for the model of the device. Information about the device state is reported by the *Status* property. When the device is ready to be used, it is available and the device status is READY.

Di	Unique identifier of the device
OwnerUCI	Device owner's UCI
Dmi	The device model
SecLevel	Security level of the device
Nickname	Nickname assigned by the owner
Status	Current status of the device

Figure 3: Device properties

UserUCI	Device user's UCI
SecLevel	Security level of the device
Nickname	Nickname assigned by the user
ValidityInterval	Period of time in which the user can use the device
Device capability	Device resource type
Usable quantity	Maximal quantity of the device capability user can use

Figure 4:Additional properties for temporary devices

A further innovative aspect of our model is that it supports the notion of *temporary device* in order to model cases in which a user borrows devices from other users. A temporary device is a device belonging to a user, referred to as *device owner*, temporary associated with another user, referred to as *device user*. Beyond the properties specified for a device, additional properties, shown in Figure 4, are associated with a temporary device. For a temporary device we store information about the temporal interval in which the user borrowing the device is authorized to use it. Moreover, the device owner can limit the use of certain device capabilities (e.g. the disk space of a laptop), by setting a usable quantity by the device borrower. For example, Alice can lend the data storage device to Bob for a month, for at most 10 megabytes. A temporary device can also be a device supplied by the Policy Engine in order to store messages not accepted by other devices. For example, Bob may wish not be disturbed, and so rejects all the messages arriving at his phone, but he wants to store a notification of the received messages in a temporary device. The device borrower may wish to use another nickname or may wish to assign a different security level instead of the ones defined by the owner. These new information are associated with the temporary device and are visible only from the borrower.

3.2 Representation of Users

Information about user characteristics is supported by means of *subject credentials*. Subject credentials assert properties about a user, either personal characteristics, or characteristics and properties deriving from relationships the user has with other users (e.g., qualification within an organization) [8]. A user can define policies for routing messages received at his/her UCI number based on the values of the credential properties associated with the message sender. Credentials are released by organizations

```
<!DOCTYPE personalRecord[
<!ELEMENT personalRecord(name,birthday,address)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT dateOfBirth EMPTY>
<!ELEMENT address (#PCDATA)>
<!ATTLIST personalRecord id ID (#REQUIRED)>
<!ATTLIST birthday      dd CDATA #REQUIRED
                        mm CDATA #REQUIRED
                        yy CDATA #REQUIRED>]>
```

Figure 5: A credential type

```
<userCard user="Alice">
  <personalRecord id="123">
    <name>Alice Brown</name>
    <birthday dd="18" mm="11" yy="1977"/>
    <address>…</address>
  </personalRecord>
  <publicLibrary id="789">
    <name>Morris Township</name>
  <publicLibrary>
</userCard>
```

Figure 6: A user card of a user Alice

authorized for such purpose (e.g., the UCI Service Bureau, governmental authorities, the company for which a user works, the public library). Any organization defines templates of credentials (called *credential types*) and uses them to generate credentials for its users. For example, the government authority specifies a template for the driver license. Figure 5 shows an example of credential type describing the personal record through an XML DTD.

A user can receive different credentials from different credential authorities. Moreover, he/she may wish to attach to a message some selected credentials, among the ones he/she holds, or to leave the message anonymous. In order to address such requirements the concept of *user card* has been introduced. A user card is a set of credentials associated with a message by the sender. The message receiver can verify such credentials in order to accept the message. Figure 6 shows the XML representation of Alice's user card. The user card contains two credentials: the first one contains her personal record, whereas the second one is released by a public library.

To facilitate the specification of rules, the definition of *user groups* is supported. A group is a set of users specified by means of *a set of UCIs.* By specifying a group name and associating with it a list of UCIs it is possible to define a new group. For example, Bob can specify the `FRIENDS` group defining the label `FRIENDS` and listing the UCI of his friends. The specification of groups allows a user to define a common behavior for the messages arriving from a set of users, instead of define the same policy for different users.

3.3 Representation of Messages

With the term "message" we mean any kind of information that can be sent from a user to others by the Internet and telecommunication networks. Therefore, a message can be a phone call, a video, an email, and a voice message. Some additional information are attached to message contents, such as: the content data format, the message size, the sensitivity level, the compression status (if applicable), the encryption status (if applicable), the signature (if applicable) and the time in which it was sent. Moreover, the sender's UCI and user card can be attached to the message. The table in Figure 7 reports all properties representing such additional information.

SUCI	Message sender's UCI
RUCI	Message receiver's UCI
Sdate	Date when the message has been sent
Size	Size of message content
MsgType	Content type
CredType	Credential type of the sender
SecLevel	Sensitivity level of the msg content
Encrypted	Encryption status of the msg content
Compressed	Compression status of the msg content
Sign	Signature of the sender

Figure 7: Message properties

Message receivers can use all those information for establishing, by means of proper rule conditions, whether to accept or reject a message.

Some of the above information is also relevant for the UCI system itself. In particular, by means of the content format property, the Policy Engine can determine which of the receiver's devices are compatible with the message. That is, the device that can directly or indirectly (by means of a translation service) handle the message. For example, a fax machine can directly handle a fax, whereas a PC can handle it indirectly, through a translation of the message contents into a gif image. Moreover, by means of the message sensitivity level, the Policy Engine can determine, among the compatible devices, the ones "secure", that is, the ones having a security level greater (or equal) to the value of the message sensitivity level.

Note that message sender may not specify some of the message properties. For example, the sender can leave the message anonymous without specifying his/her UCI or can leave the message without credentials when he/she does not wish to release his/her personal information to the receiver.

Even though our current system only supports for messages the properties reported in Figure 7, it is possible to introduce additional properties by properly refining the message type hierarchy. New message subtypes can be defined containing all required additional properties starting from the message type currently provided by our system.

4 The Policy Rule Language

The Policy Rule Language we propose is based on the ECA (Event, Condition, Action) paradigm, typical of active databases [7], and addresses all requirements outlined in Section 2. The language allows users to specify rules stating policies for message routing, re-routing, and notification based on the user preferences. The preferences stated by a user as a set of rules are associated by the Policy Engine with the UCI of the user and are stored into the *UCI user profile*. The general format of a UCI rule is as follows. In the notation used here and in what follows, square brackets denote optional components.

UCI Rule Format

RuleName:
ON *Event*
[WHENEVER *Condition*]
Action
[*ValidityInterval*]

As it can be seen, a rule consists of several components. The *RuleName* component represents the rule identifier. UCI users can retrieve rules by means of such identifier for visualizing, dropping or modifying them. The *Event* component represents the event upon which the rule is *triggered*. *Condition* is an optional conditional expression, defined by means of the condition language we have developed, specifying a filter on the applicability of the rule. *Action* is an expression specifying the devices where the message should be routed or a notification message should be sent. *ValidityInterval* is a temporal condition, also expressed by means of the condition language we developed, representing the time period during which the rule is enabled. For example, the interval `[DATE BETWEEN NOW TO FOREVER]` specifies that a rule is enabled forever since it is entered into the Engine database. By contrast, `[DATE BETWEEN 01:20:2002 TO 07:31:2002]` specifies that a rule is enabled from January the 20th, 2002 until July the 31st, 2002. If *ValidityInterval* is not specified we assume the rule is always enabled. By means of the *ValidityInterval* it is possible to state that certain rules are not always enabled; rather, they are enabled only during specific temporal intervals. Such a feature, which is not provided by conventional database triggers, is quite useful whenever users need to plan or modify in advance their routing rules.

The semantics of a UCI rule is based on semantics of the ECA paradigm. Such semantics states that the routing *Action* specified in the rule named *RuleName* is executed upon the occurrence of event *Event* whenever the condition *Condition* is verified and the rule is enabled. A rule is enabled if the time, in which the rule condition is checked, is contained in the validity interval *ValidityInterval* specified for the rule.

In the remainder of the section we discuss in more details the policy rule language, whose grammar is shown in Figure 14 in Appendix A. In particular, in Subsection 4.1 we discuss rule events, whereas in Subsection 4.2 we discuss the condition language. Finally, in Subsection 4.3, we discuss the action specification language.

4.1 Events

Our rule language is essentially based on events of two types: a message arrival at *UCI* number; a message arrival at device *di*. By means of these kinds of event it is possible to specify policies both for messages arriving at a UCI and for messages routed to a device that should be re-routed to another device.

Rules specified at UCI level allow the UCI user to specify "general preferences" on messages arriving at his/her UCI, or for which a notification message should be generated. For example, he/she can specify conditions on the user characteristics, on the arrival time, on the group the sender belongs to. These conditions are independent from the specific devices that will handle the message. By contrast, rules specified at device level allow the UCI user to state "specific preferences" on messages arriving at a

particular device. For example, the user can specify that an email containing huge image arriving at the laptop should be re-routed to the PC and a notification message sent to the mobile phone, or a message arriving at the laptop containing a video should be re-routed to the PC, whenever the disk space is not enough to contain it.

4.2 Conditions

The condition language we have developed is quite rich. It allows one to specify basic conditions on all properties characterizing devices, messages, and users. The language also supports the specification of a large variety of temporal conditions. Those basic conditions can be combined together by boolean operators in order to specify complex conditions.

An important aspect of our approach is that the condition language can be uniformly used for various purposes. In particular, it can be used for filtering messages arriving at an UCI or at a device, and for selecting devices. In the next section, we discuss how the language can be used for device selection. Moreover, the language provides a large variety of special-purpose predicates. In particular, the language provides several predicates for testing the device status, such as the `NOTAVAILABLE` and `BUSY` predicates, as well as the predicates for testing sender information, such as whether the message is anonymous. Several predicates are also provided concerning the message itself, such as predicates to test the sensitivity of the message.

Example 1 *The following expressions are examples of conditions on devices:*

- `STATUS IS BUSY`. *The condition is verified if the device (against which the condition is checked) is busy.*
- `DEVICE PROPERTY type = PHONE`. *The condition is verified if the device is of type phone.*
- `DEVICE PROPERTY displaySize > 3`. *The condition is verified if the device has a display with more than 3 lines.*
- `BORROWEDDEVICE`. *The condition is verified if the device is borrowed by another user.*

By contrast, the following expressions are examples of conditions on users, temporal information, and message contents, respectively:

- `SENDER HAS UCI(100)`. *The condition is verified if the message sender has UCI 100.*
- `ON SUNDAY`. *The condition is verified if the day of the week in which the message arrives is Sunday.*
- `TIME BETWEEN 5pm AND 7pm`. *The condition is verified if the message arrival time is between 5pm and 7pm.*
- `MSG IS NOT ENCRYPTED`. The condition is verified if the message content is not encrypted. □

4.3 Actions

By means of the routing action component of a rule it is possible to specify the actions to be performed upon a message arrival at UCI or a specific device. Our language supports three action types, namely message rejection, routing, and notification.

Some key aspects of our action language have to be noted. The first is that message re-routing, that is, routing a message to a device, and from this to another one and so on, can be simply achieved in our language by specifying several rules, containing one or more of the three action types provided by our action language. Thus, message re-routing does not require the introduction of additional action types. A first rule will be typically associated with the UCI and will have the effect of specifying an initial device to which the message has to be routed. A second rule associated with that device will specify a routing action having the effect of re-routing this message to another device. Such device in turn may have specific rules associated with it to perform further routing actions. Our approach is thus orthogonal and very simple to use. It has the drawbacks that rule executions may result in non-terminating routing loops. The approach we take to handle such a problem is briefly discussed in Section 5.

The second key aspect is that the action component of a rule may contain both routing actions and notification actions. Therefore, a user can require that a message be handled by a given device and, at the same time, that a notification be sent to another device. The notification is a new message to be routed to devices belonging to the UCI owner. The notification may contain different kinds of information, such as the sender credential, the message type, the arrival time. The notification format is independent from the device that will handle it. The Policy Engine, through specific translation and adaptation services, adapts the message to the device will handle it. For example, if the notification has to be sent to a mailbox, the Policy Engine will generate an email message. By contrast, if the notification has to be sent to a mobile phone, the Policy Engine will generate a notification in form of a speech or an SMS.

A third key aspect of our action language is that we support message routing to a single device or to multiple devices. We refer to the former as “unicast routing”, whereas to the latter as “multicast routing”. Under unicast routing the message is actually routed to a single device, which in turn may re-route it to other devices. Under multicast routing, the message is directly sent by a single rule action to multiple devices. Multicast routing actions in our language have two different forms, corresponding to two different formats in the specification of the devices to which the message has to be routed. Such devices can be specified as:

```
WORKINGTIMECALLS:
   ON ARRIVAL AT UCI(100)
   WHENEVER TIME BETWEEN 9HH TO 17HH
   ROUTE TO phoneAtOffice;
FREETIMECALLS:
   ON ARRIVAL AT UCI(100)
   WHENEVER TIME BETWEEN 17HH TO 9HH
   ROUTE TO phoneAtHome;
NOTATHOME:
   ON ARRIVAL AT DEVICE(phoneAtHome)
   WHENEVER STATUS IS NOTANSWERING
   ROUTE TO mobilePhone;
```

Figure 8: Rules of Example 2

- *A sequence*. The rule specifies a sequence of device identifiers, where the message must be sent (the order in which he/she specifies the devices determine the relevance of the device). In this case, we assume two possible interpretations of the rule. The first interpretation is "*the message has to be sent to all the specified devices"*. This interpretation is specified by the keyword `ALL`. "*The message has to be sent to at least one of the specified devices"* is the second interpretation. Thus, in this interpretation, when a device handles the message we do not have to consider the remainder of the devices. This interpretation is specified by the keyword `ANY`, and `ANY` is the default.
- *A set defined by a condition*. The rule specifies a condition on device capabilities or states. All devices verifying the condition are included in the set of message recipients. The possibility of specifying devices based on their capabilities or states increases the portability of a rule. Because the rule selects devices based on their capabilities instead of identifiers, the rule is still applicable even if some devices are dropped. Also, in this case, the two different interpretations previously introduced can be applied.

By combing all the above options, the resulting action language is quite expressive, even though it is based on a few simple concepts. By an orthogonal combination of several rules, one may express complex and articulated routing policies.

4.4 Illustrative Examples

In the following we introduce some examples of policy rules, based on the scenario presented in Figure 1. For sake of simplicity, in the examples, we consider the rules always enabled.

Example 2 *Suppose Bob wishes to route all phone calls to his work phone during working time, and to his home phone at any other time, and if his home phone does not answer, calls have to be re-routed to his mobile phone. Rules in Figure 8 specify those routing policies.* `phoneAtHome`, `phoneAtOffice`, `mobilePhone` *are the nicknames Bob uses for the corresponding devices.* □

```
RELEVANTFAX:
  ON ARRIVAL AT UCI(200)
  WHENEVER MSG CONTENT TYPE IS FAX
  ROUTE TO fax; printer USING FAX2PS;
PRINTINGFAX:
  ON ARRIVAL AT DEVICE(fax)
  NOTIFY laptop;
LAPTOPNOTAVAILABLE:
  ON ARRIVAL AT DEVICE(laptop)
  WHENEVER STATUS IS NOTAVAILABLE
  REJECT; NOTIFY myPhone;
VIPSCALLS:
  ON ARRIVAL AT UCI(200)
  WHENEVER SENDER IS IN GROUP VIPS
  AND MSG CONTENT TYPE IS PHONECALL
  ROUTE TO myPhone;
```

Figure 9: Rules of Example 3

In the example, Rule `WORKINGTIMECALLS` specifies that a message, whose arriving time is between 9 a.m. to 5 p.m., has to be routed to the office phone, whereas, Rule `FREETIMECALLS` specifies that a message, whose arriving time is between 5 p.m. to 9 a.m., has to be routed to the residential phone. Rule `NOTATHOME` specifies that if the residential phone does not answer, the message has to be re-routed to the mobile phone. Next example shows some more articulated policies, including rules requiring notification messages and the translation of message contents.

Example 3 *Suppose Alice wishes to route a fax to the fax machine or, if the fax machine is not available, to the printer translating the fax into a postscript file. Moreover, she wishes to be notified on the laptop whenever the fax machine has printed out the fax. If the laptop is not available a message routed to it must be rejected and a notification message sent to her home phone. Finally, she wishes to receive all the phone calls from the* `VIPS` *group at her home phone. The rules in Figure 9 specify those routing policies.* `myPhone`, `printer`, `laptop`, `fax` *are the nicknames Alice uses for the corresponding devices.* □

In the example, Rule `RELEVANTFAX` specifies that a message, whose content is of type FAX, arriving at Alice's UCI should be routed to device identified by 212-555-111. If such device is not able to handle such message, the message should be translated into a postscript file (using a fax to postscript translation service) and sent to device identified by 203.1.3.3. The routing action component of the rule is based on the `ANY` interpretation presented above. Rule `PRINTINGFAX` specifies that whenever a message arrives at device identified by 212-555-111, the device should handle it and a notification message should be sent to device identified by 203.2.1.2. Rule `LAPTOPNOT-AVAILABLE` specifies that whenever device identified by 203.2.1.2 is not available messages routed to it should be rejected and a notification message sent to her phone.

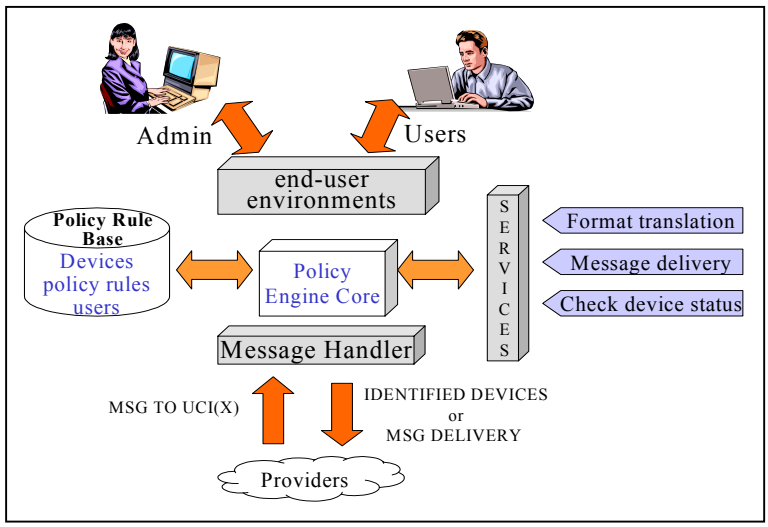

Figure 10: Policy Engine Architecture

Finally, Rule `VIPSCALLS` specifies that messages arriving at Alice's UCI should be routed to her phone (identified by 973-321-123) whenever they are phone calls and the caller belongs to group `VIPS`.

5 The Policy Engine Architecture

The Policy Engine System has been developed to implement the policy rule language and to provide environments for users that have to configure, use and monitor the rules. The overall system architecture, graphically illustrated in Figure 10, consists of a number of components. Among those components, the Policy Engine Core represents the kernel of the system since it is in charge of message dispatching. Message dispatching is executed according to the policy rules stored in the Policy Rule Base, a large database, implemented on top of a commercial DBMS, storing all UCI rules and information about users and devices. The Policy Engine Core thus implements the algorithms for the evaluation and execution of such rules. The Policy Engine is also equipped with a set of facilities for the interaction with UCI users, UCI administrators, and service providers. Tailored end-user environments have thus been developed for helping the UCI administrators and UCI users to interact with the system. Moreover, facilities for receiving message delivery requests and for adding/removing services supplied by external providers have been integrated in the Policy Engine.

In the remainder of the section we first describe in Subsection 5.1 the steps according to which the rule evaluation process is organized. We then present in Subsection 5.2 facilities supporting the user interaction with the Policy Engine. Some details of the implementation and of the message delivery time are reported in Section 5.3.

5.1 Rule Evaluation Process

The rule evaluation process is organized according to three main steps. Let *U* be a user receiving at his UCI *UCI* a message *m*. The following steps are performed in order to deliver the message *m*:

(1) selection of the devices of *U* compatible with *m*;

(2) selection of the applicable rules, among those associated with *UCI*;

(3) evaluation of the routing actions specified by the rules selected at step (2).

In the remainder of the section we describe these steps. The rule evaluation process has several "critical issues", such as for example rule execution termination. At the end of the section we briefly discuss such issues and how the Policy Engine addresses them.

Selection of compatible devices. A user can have, as owner or borrower, several devices with similar capabilities that can handle different kinds of messages. Therefore, the Policy Engine selects among the devices, belonging to *U,* those compatible with the message *m*. A device *d* is compatible with a message *m* if the following two conditions are verified:

- *d* can handle directly or indirectly *m*. A device can handle directly a message when it supports the message content type. When the device cannot directly handle the message *m* and more than one translation service can be used for *m*, then the Policy Engine chooses "the best one", that is, the translation service with the lower cost or the one having higher priority based on the order established by the Policy Engine Administrator.
- The sensitivity level of the content of message *m* is at least equal to the security level of the device.

The set of compatible devices can result in an empty set when no device can handle the message or none of them is secure enough for the message. In this case a notification message is generated containing this error description and sent to the user. Moreover, the original message is rejected or temporally stored in the Policy Engine devices.

```
TOOFFICEPHONE:
   ON ARRIVAL AT DEVICE(phoneAtHome)
   ROUTE TO phoneAtOffice;
TORESIDENTIALPHONE:
   ON ARRIVAL AT DEVICE(phoneAtOffice)
   ROUTE TO phoneAtHome;
```

(a)

```
OUTOFOFFICECALLS:
   ON ARRIVAL AT UCI(phoneAtHome)
   WHENEVER TIME IN WEEKEND
   ROUTE TO phoneAtOffice;
LUNCHTIMECALLS:
   ON ARRIVAL AT DEVICE(phoneAtOffice)
   WHENEVER TIME BETWEEN 12HH TO 14HH
    ROUTE TO phoneAtHome;
```

(b)

Figure 11: Examples of rules generating loops

Selection of applicable rules. After the selection of the compatible devices, the Policy Engine selects the rules to be executed by applying the following filters according to the order specified below:

(1) Select only the enabled rules, that is, those rules for which the message sending time falls within the validity interval of the rule.

(2) Among the rules selected at step (1) determine those for which the corresponding event has occurred.

(3) Among the rules selected at step (2) determine those for which the condition is true.

If the above selection process returns more than one rule, the Policy Engine first orders the rules according the rule priorities. The priority adopted by the Policy Engine is determined by the order according to which the rules have been listed in the user profiles stored in the Engine Database. Then, the Policy Engine evaluates, rule by rule, their actions until one of them returns at least a device that can handle the message. If at least one rule has an action able to deliver the message to at least a device the message is considered delivered with no errors. Otherwise, a notification error is returned to the user.

If no rule is selected after applying the above filters, it means that no rule has been specified for such message, and then the "default behavior" is applied. The "default behavior" represents the behavior followed by the Policy Engine whenever no rule has been specified by the user or no rule specified by the user can be applied on such message. In these situations, the Policy Engine tries to deliver the message on each of the compatible devices until one of them handles the message. When the first device handles the message the process terminates. In other words, the Policy Engine applies the `ANY` policy as default behavior. If no device is able to handle the message, an error notification message is sent to the user.

Evaluation of routing actions. Whenever the condition of a rule is verified, the Policy Engine evaluates and possibly executes the action expression specified in the rule. If the routing specification requires a message rejection, the Policy Engine simply rejects it and the process terminates. Otherwise, if the routing specification requires a message routing, the Policy Engine evaluates the routing specification against the compatible devices in order to find the devices that can handle the message. If the routing specification requires the translation of the message into another format such translation is performed. Then, the message is routed to such devices according to the `ANY` or `ALL` policy. Under the `ANY` policy, the delivery process terminates when the first device specified in the routing specification handles the message. By contrast, under the `ALL` policy, the process terminates when all the specified devices handle the message. If the notification specification is present, a notification message is generated and the notification specification evaluated in order to determine the devices where the notification message should be routed.

Critical issues. The richness of our language and the presence of possibly large sets of rules introduce several critical issues concerning the rule execution system. Here we discuss some of those issues and outline the solutions we have adopted.

A first issue is related to the presence of loops in rule execution as stated from the following example.

Example 4 *Suppose Bob defines the rules in Figure* 11(a). *Then, whenever a message is routed to the phone at home or to the phone at office it cannot be delivered because the evaluation of the rules never terminates. By contrast, suppose Bob defines the rules in Figure* 11(b). *During the weekend, between noon and 2pm, the behavior of these rules is the same of the previous one. However, at any other time the message is delivered to one of the two devices* □

From the previous examples we can point out two kinds of loops that can arise: *permanent* and *conditional*. *Permanent loops* arise when the evaluation of a set of rules causes a loop independently from the time in which the Policy Engine evaluates the rules in the set. By contrast, *conditional loops* arise when the evaluation of a set of rules causes a loop only when a particular combination of conditions is verified.

The current solution we have implemented in the Policy Engine to address such problem is based on counting the number of times a message *m* is re-routed to the same device. Whenever this number overcomes a given threshold (called *maximum number of loops*) the Policy Engine stores the message into the "parking queue" for a period of time (called *pickup delay time*). After the pickup delay time, the Policy Engine tries to re-evaluate the rules in order to verify whether the situation is changed (i.e. the evaluation of the rules that caused the loop is changed) and the message can thus be delivered. If the situation has changed, the Policy Engine delivers the message, otherwise it stores again the message into the queue. A message can be stored in the queue for at most *TTL* (time to live) times. After TTL times the message is rejected and a notification message sent to

the UCI user. The maximal number of loops, the pickup delay time, and the TTL are parameters that the Policy Engine Administrator can set by means of the facilities the Policy Engine offers. Other issues concerning rule evaluation are related to:

- Complex conditions intrinsically inconsistent. For example, the condition "`MSG ARRIVAL TIME BETWEEN 12pm AND 2pm AND MSG ARRIVAL TIME NOT BETWEEN 12pm AND 2pm`", or the condition "`MSG CONTENT TYPE = image AND MSG CONTENT TYPE ≠ image`".
- Rule validity periods that do not match with the temporal conditions expressed in the rules. For example, a rule such as "messages arriving during working week should be routed to my phones at office" that is valid during the weekend.

Such kind of rules can never be triggered. The Policy Engine addresses these situations by monitoring the application of rules defined by the user for a period of time. If the Policy Engine has never executed a rule during such period of time (for example a week), the Policy Engine creates a notification message containing such rule and sends it to the UCI of such user. In this way the UCI owner can check his/her rules and verify the presence of mistakes.

5.2 Policy Engine Facilities

Several facilities have been developed for allowing the users, administrator, and external providers to communicate with the Policy Engine. In the remainder of the section we briefly outline such facilities.

Facilities for the Policy Engine Administrator. By means of a Web interface the Policy Engine Administrator can access the Policy Engine through a browser. A graphical interface allows the Administrator to add, remove, and update user profiles. Each profile contains the user UCI, credentials, routing rules, and devices belonging to a user. Whenever the Administrator creates a new user, a user profile, containing the new UCI and the password, is generated. After that, the Administrator, as a credential authority does, assigns credentials to the user and certifies their truth. In this way message receivers are guaranteed of the truth of the information contained in the sender's user card.

Other facilities have been developed in order to display and set a number of parameters of the Policy Engine (e.g., the maximum number of loops, the pickup delay time, the TTL parameters etc.). Moreover, facilities have been developed for the management of the Policy Engine devices. These facilities allows the Administrator to handle (add, remove, show) its devices and to rent/lend them to its users. Finally, facilities have been developed to add/remove/update device type, model, credential types, services for translating the message formats, checking the device states, and delivering messages to receivers.

Facilities for the Policy Engine User. By means of a graphical interface the Policy Engine User can perform different actions on his/her profile. New devices can be added, removed and/or updated establishing nicknames and security levels. Moreover, the user can lend his/her devices to other users for a period of time. The most relevant feature of the user environment is, however, the rule definition tool. By means of this tool the user is driven by the system through the formulation of a rule and does not need to have any knowledge concerning the policy language. The tool takes care of generating the proper rules based on the preferences graphically selected by the user. Figure 13 in Appendix shows a snapshot of the routing rule definition interface allowing a user to define a condition of a rule.

Facilities have also been developed allowing the user to define/update/show groups of users. Moreover, he/she can define user cards to be attached to a message. The user card contains credentials received by the Policy Engine Administrator. A "default user card" can be specified, that is, a user card always attached to a message, if no other user card is specified. Finally, facilities have been developed for defining/updating and changing rule priorities.

Message handling facilities. Given a request of sending a message *m* to a given *UCI*, the Policy Engine can perform different actions depending the services it has available. We recall here that the Policy Engine can be coupled with external services provided by the telecommunication network software or by other providers. If the Policy Engine is equipped with services for checking the device states and for delivering message to devices, the Policy Engine works as a switcher. That is, given a request the Policy Engine evaluates the receiver's rules, checks the device states and delivers the message to the selected devices. If those services cannot be integrated in the Policy Engine, the Policy Engine can be used as a "selector" of devices to which a message could be routed. Thus, the provider requesting to send *m* to *UCI*, receives back from the Policy Engine a list of devices to which it can try to send the message and the list of devices that should be notified. If the provider is not able to deliver the message to the devices selected, then it asks again the Policy Engine for new devices. This process is performed until a device is found that handles the message or no device is found and the message is rejected.

5.3 Implementation and Device Identification Time

A prototype of the Policy Engine has been implemented in Java on top of the Oracle 8i object-relational DBMS (version 8.1.7). The DBMS in particular is used for storing all information about users, devices, and policy rules.

In evaluating the performance of the system, we focused on the operation of device identification, since this is the most critical operation. Such operation consists in accessing the user profile stored in the Engine Database, evaluating the policy rules and returning the device(s) where the message should be routed. Therefore, we ran a number of experiments measuring the time it takes to determine the device(s) to which a message has to be delivered upon the

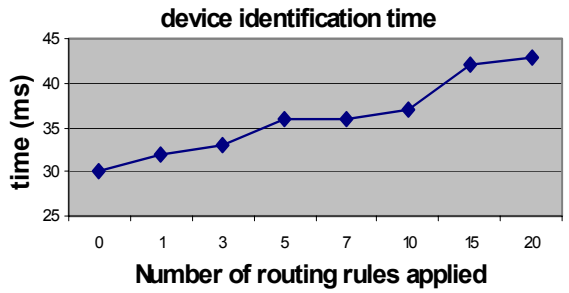

Figure 12: Performance results

message arrival at the UCI engine (such time is called *device identification time*). The experiments have been carried out for varying values of the number of rules associated with a UCI. The machine we used in the experiments is a Pentium III, 1 GHz, 256 Mbytes RAM. In the experiments we made the assumptions that no translation is performed on the message, and that the message is never stored in the parking queue. Figure 12 shows the performance results. The device identification time linearly increases with the increase in the number of rules. If no rule applies to the message, the device identification time is around 30 ms. In such time interval, the Policy Engine has to load the user profile and to check that no rule applies to the message. By contrast, if 20 rules are evaluated for determining the devices that will handle, the message the device identification time increases to 43 ms. The application of 20 rules, however, is really an extreme case. In real scenarios the number of rules that can apply to a message is around 5-7. Note, also, that the performance results of our system falls within the expected times for such kind of operations for commercial systems [1]. Such results are really encouraging if we also take into account the machine used for the experiments and the language used for the implementation. However, issues related to scalability and performance in real systems, mentioned in [1], should be addressed in order to make our approach viable for real systems. Therefore, we plan to investigate in the near future issues related to data replication and partitioning, common in current voice telecommunication architectures.

6 Conclusions and Future Work

In this paper we have presented a declarative rule language for expressing routing policies for a Unified Communication Identifier system. The language is characterized by a number of features. It supports both unicast message routing, according to which a given message is routed to a single device, and multicast message routing, allowing one to route a message to several devices. The language also provides a large variety of predicates, forming the basis for a rich condition language, by using which articulated routing policies can be specified. A Policy Engine System implementing the language and providing a number of end-user facilities has been developed.

The Policy Engine has been integrated in a Telcordia research project for services over a converged voice and data network. A key feature of the services is user control of routing rules and user data. In such project, our Policy Engine allows the identification of a phone or other device where a user can be reached based on preferences he/she states and/or the capability of the device and network.

We are now working on several research issues related to the rule language and to the system. A first research direction concerns the development of static rule analysis techniques to perform termination analysis and to check rule consistency. A second research direction concerns an extensive investigation of rule storage strategies. In the current prototype, all rules associated with the same UCI are simply stored according to a streamed representation into the database. Therefore, fetching all rules associated with a UCI requires accessing a single, even though very large, tuple. In the current version all rules associated with a UCI are fetched, even those that are not enabled. We thus plan to evaluate alternative strategies to improve the performance of the rule selection process. A third direction research deals with developing an XML-based model of our rule language to enhance interoperability. A fourth research direction concerns with the development of a comprehensive authorization model for the Policy Engine System and to investigate security and dependability of the proposed system. Finally, as already mentioned in the previous section, we will investigate scalability and performance of our approach.

Acknowledgements

We wish to thank the master students Giuseppe Garau, Valeria Galli, and Stefano Franzoni that helped us in the development and implementation of the system and in gathering experimental results.

References

[1] M. Cochinwala. Database Performance for Next Generation Telecommunications. In Proc. of ICDE, Germany, 2001.

[2] M.Cochinwala, H.Hauser, and N.Suri. Network Convergence using Universal Numbers: The UPT Project. *In Proc. of VLDB workshop on database on telecommunications*. Rome, September , 2001.

[3] M. Foster, T. McGarry, and J. Yu. Number Portability in the GSTN: An Overview. Internet Draft, Feb. 2001.

[4] G.C. Kessler, and P.V. Southwick. ISDN: Concepts, Facilities, and Services. McGraw-Hill. 1998.

[5] International Engineering Consortium. Unified Messaging. Tutorial. http://www.iec.org/.

[6] S. Pollock. A Rule-Based Message Filtering System. *Transaction on Office Information System*, 6(3) July 1988.

[7] J. Widom and S. Ceri. Active Database Systems. Morgan Kaufmann Publisher, 1996.

[8] M. Winslett, N. Ching, V. Jones, and I. Slepchin. Using Digital Credentials on the World Wide Web. *Journal of Computer Security*, 7, 1997.

[9] W3C. CC/PP profile. Working draft. http://www.w3.org/

Appendix A: Grammar of the policy rule language

Figure 14 presents the grammar of the policy rule language. *PropertyName* is the name of a property specified for an actor of the system, whereas *Value* is a valid value for such property. Moreover, *GroupName* is the name of a group, whereas *UCI* is a UCI associated with a Policy Engine User. Finally, *Time* and *Date* are temporal expressions, respectively, on time and date.

Appendix B: Traces of rule executions

The Policy Engine prototype is equipped with a tool to trace the behaviour of the Policy Engine upon receiving a message. Figure 15(a) shows the behaviour of the system when a fax is sent to Alice from Bob and all the devices are available (we refer to the rules of Figure 9). As the reader can see, the system evaluates the compatible devices, executes the rule `RELEVANTFAX`, establishes that the message should be sent to the fax or the printer with the `ANY` interpretation and, in the second case, the message

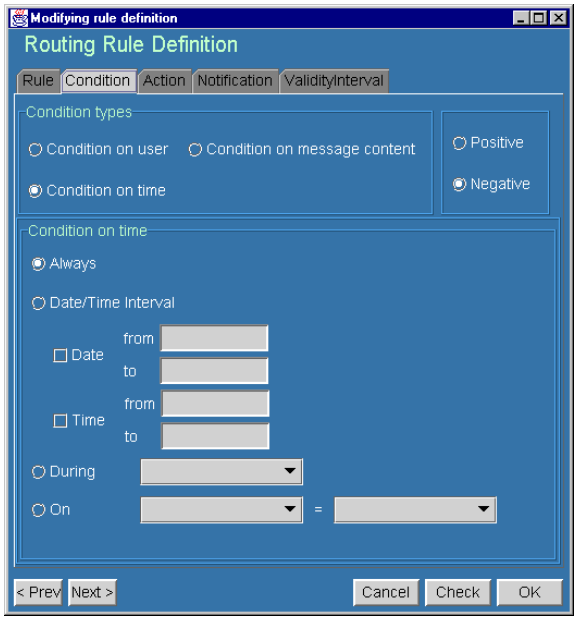

Figure 13: Snapshot of the rule definition interface

PolicyRule	::=	*RuleName*: ON *Event* [WHENEVER *Condition*] *Action* [*ValidityInterval*]
Event	::=	MESSAGE ARRIVAL AT UCI(*UCI*) | MESSAGE ARRIVAL AT DEVICE(*di*)
Action	::=	[*PolicyExpr*] [*NotificationExpr*]
PolicyExpr	::=	REJECT; | ROUTE TO [ALL | ANY] *DeviceExpr*;
NotificationExpr	::=	NOTIFY [ALL | ANY] [*DeviceExpr*;]
DeviceExpr	::=	*di* [USING *MediaSpec*] | *CondOnDevice* | *DeviceExpr*; *DeviceExpr*
MediaSpec	::=	FAX2IMG | FAX2PS | VOICE2MAIL | MAIL2SMS
ValidityInterval	::=	[*CondOnTime*]
Condition	::=	*CondOnDevice* | *CondOnMsg* | *CondOnUser* | *CondOnTime* | (*Condition*) | NOT (*Condition*) | *Condition* AND *Condition* | *Condition* OR *Condition*
CondOnDevice	::=	STATUS IS *DeviceStatus* | DEVICE SECURITY LEVEL IS *SecLevel* | DEVICE PROPERTY *PropertyName Op Value* | OWNEDDEVICE | BORROWEDDEVICE
DeviceStatus	::=	NOTAVAILABLE | BUSY | LOWBATTERY | LOWMEMORY | LOWDISKSPACE | NOTANSWERING
CondOnUser	::=	SENDER HAS [NOT] UCI(*UCI*) | SENDER IS [NOT] ANONYMOUS SENDER PROPERTY *PropertyName Op Value* | SENDER IS [NOT] IN GROUP(*GroupName*)
CondOnTime	::=	TIME BETWEEN *TimeExpr* TO *TimeExpr* | DATE BETWEEN *DateExpr* TO *DateExpr* | TIME IN WEEKEND | TIME IN WORKINGWEEK | ON [*Day* | *Month* | *Year*]
Day	::=	MONDAY | ... | SUNDAY
Month	::=	JANUARY | ... | DECEMBER
Year	::=	Integer
TimeExpr	::=	NOW | FOREVER | *Time*
DateExpr	::=	NOW | FOREVER | *Date*
CondOnMsg	::=	MSG IS [NOT] ENCRYPTED | MSG IS [NOT] COMPRESSED | MSG IS [NOT] SIGNED | MSG CONTENT TYPE IS [NOT] (*TypeName*) | MSG SENSITIVITY LEVEL IS *SecLevel*
TypeName	::=	TXT | IMAGE | HTML | SOUND | PHONECALL
SecLevel	::=	HIGH | NORMAL | LOW
Op	::=	= | ≠ | < | ≤ | > | ≥

Figure 14: Grammar of the policy rule language

```
START of message handling
   Message ID = 1013798205146.0.bob.4
   From user 'Bob' to user 'Alice'
   List of compatible devices with the message
      Device 212-555-111; translation NONE
      Device 203.2.1.2; translation FAX2PS
      Device 203.1.3.3; translation FAX2PS
      Device 203.1.2.3; translation FAX2PS
   Evaluating and executing rule 'RELEVANTFAX'
      The message will be sent to ANY of the devices
        Device 212-555-111; translation NONE
        Device 203.1.3.3; translation FAX2PS
      Found routing rule on device 212-555-111
      Evaluating and executing rule 'PRINTINGFAX'
         Message will be sent to device 212-555-111
         Message has been sent to device 212-555-111
         Notification will be sent to ANY OF the devices
            Device 203.2.1.2; translation NONE
         Sending message to the device 203.2.1.2
         Message has been sent to device 203.2.1.2
      End of rule 'PRINTINGFAX'
   End of rule 'RELEVANTFAX'
END of message handling
```

(a)

```
START of message handling
   Message ID = 1013798692667.0.bob.4
   From user 'Bob' to user 'Alice'
   List of compatible devices with the message
   … as above
   Evaluating and executing rule 'RELEVANTFAX'
      The message will be sent to ANY of the devices
         Device 212-555-111; translation NONE
         Device 203.1.3.3; translation FAX2PS
      Sending message to the device 212-555-111
      Message NOT delivered to device 212-555-111
      Sending message to the device 203.1.3.3
      Message has been delivered to
            device 203.1.3.3: translation FAX2PS
   End of rule 'RELEVANTFAX'
END of message handling
```

(b)

Figure 15: Trace # 1

should be translated into a postscript file. The rule PRINTINGFAX is evaluated. Since the rule execution terminates without errors, the system does not deliver the message to the printer. A notification message is sent to the laptop.

By contrast, Figure 15(b) shows the behaviour of the system when the fax machine is not available. Till the evaluation of the PRINTINGFAX rule, the behaviour of the system is the same. When the system evaluates such rule, since the fax is not available, the message cannot be delivery to it. Therefore, the system tries to deliver the message to the printer after translating it into a postscript file. The operation succeeds, thus the process terminates. Note that, in such situation no notification message has been sent to the laptop.

Consider now the situation in which Alice adds the rule:

```
PRINTERNOTAVAILABLE:
  ON ARRIVAL AT DEVICE(printer)
  WHENEVER STATUS IS NOTAVAILABLE
  ROUTE TO fax;
```

Suppose also that the fax machine and printer are not available and the Policy Engine Administrator sets the TTL parameter to 2 and the maximum number of loops to 2. The Policy Engine tries to send the fax to the two devices. Since they are not available, the system starts an error handling section in which it establishes that the message should be re-routed. The process starts again but, also in this case, the message is not delivered to one of the devices. Therefore, the system stores the message in a temporary storage device. After the pickup delay time, the system tries to deliver again the message. If one of the devices becomes available the message is delivered. Otherwise, if the message is still undelivered (following a process similar to the one described), the message is rejected and a notification sent to Alice.

```
START of message handling
   Message ID = 1013799307191.0.bob.4
   From user 'Bob' to user 'Alice'
   List of compatible devices with the message
      Device 212-555-111; translation NONE
      Device 203.2.1.2; translation FAX2PS
      Device 203.1.3.3; translation FAX2PS
      Device 203.1.2.1; translation FAX2PS
      Device 203.1.2.3; translation FAX2PS
   Evaluating and executing rule 'RELEVANTFAX'
      Message will be sent to ANY of the devices
         Device 212-555-111; translation NONE
         Device 203.1.3.3; translation FAX2PS
      Sending the message to the device 212-555-111
      Message NOT delivered to device 212-555-111
      Sending the message to the device 203.1.3.3
      Evaluating and executing rule 'PRINTERNOTAVAILABLE'
         Message will be sent to ANY of the devices
            Device 212-555-111; translation NONE
         Sending the message to the device 212-555-111
         Message NOT delivered to device 212-555-111
         Notification skipped for rule 'PRINTERNOTAVAILABLE'
      End of rule 'PRINTERNOTAVAILABLE'
      Notification skipped for rule 'RELEVANTFAX'
   End of rule 'RELEVANTFAX'
   START of error handling
      Message will be rerouted
   END of error handling
END of message handling
START of message handling
   Message ID = 1013799307191.0.bob.4
    As above (a)
   START of error handling
      Message has been stored into the database
      A notification will be sent to the user
   END of error handling
END of message handling
```

(a)

Figure 16: Trace # 2