Chapter 15

Policies for Feature Interaction Resolution

Magdi Amer[1], Ahmed Karmouch[1], Tom Gray[2], Serge Mankovskii[2]
[1] Multimedia & Mobile Agent Research Laboratory, Dept. of Electrical and Computer Engineering, University of Ottawa, Ottawa, Ontario, Canada K1N 6N5, http://deneb.genie.uottawa.ca/
[2] Mitel Corporation, 350 Legget Drive, Kanata, Ontario, Canada K2K 2W7, http://www.mitel.com/

Key words: Policies, agent, fuzzy logic, conflict detection and conflict resolution.

Abstract: Telephone systems are marked by the provision of many hundreds of features. Conflict between these features is inherent as the actions of one feature can be in direct opposition to the aims of another feature. Most telecommunication service providers resolve the feature interaction problem by providing specific instructions in their management software. This approach suffers from the complexity of the resulting code and the difficulty of adding new features to the system. In this paper, we propose an agent-based architecture in which the actions of each agent are controlled by a set of policies. We also introduced the concept of fuzzy-policies, which are policies whose suitability for handling an event is calculated dynamically, based on the value of some fuzzy-variables. Conflicts are resolved using an arbitrator agent, which recalculates the suitability of the proposed actions of each agent and deduces the best action that satisfies the end user. The end user has the ability to add new policies, or modify the values of the fuzzy-parameters of the user-agent to alter the behavior of the system, thus obtaining a more personalized service.

1. INTRODUCTION

Telephone systems are marked by the provision of many hundreds of features that can be used to select the most appropriate way to dispose of a call for a user. Conflict is inherent in the provision of multiple features for a user. The actions of one feature can be in direct opposition to the aims of

J. N. de Souza et al. (eds.), *Managing QoS in Multimedia Networks and Services*

another feature. As a simple example the actions of the features Call Forward Busy and Call Waiting are usually given. With Call Forward Busy, a call to a user talking on the telephone will be directed to another telephone. In Call Waiting, the incoming call will be announce with a tone to the user who has an option of taking it directly while the incoming caller continues to hear audible ringing. These two features cannot be simultaneously active since they will direct incompatible outcomes for a call. With many hundreds of features, there is an explosion of such interactions, which must be detected and resolved.

The coming converged voice/data marketplace will create opportunities for many new features. As can be expected this will only make more difficult the problem of feature interaction since features can now depend on parameters from the data as well as the voice world. In addition, these new features will more than in the past depend on the current user context and so will become much more dynamic than in the past. Most current telecommunication service-providers resolve the feature interaction problem by providing specific instructions in their management software to handle scenarios where feature interaction may occur. This increases the complexity of the resulting code and the difficulty of adding new features to the system. Moreover, the output of the resolution between conflicting features is predefined.

This motivated several researches to be conducted using a multi-agents architecture for feature interaction resolution [Buh98 and Wei97]. These architectures proved to be more flexible and extensible than classical systems, but they still suffered from the fact that the output of the resolution between conflicting features is predefined. Thus, the end-user did not have a means of affecting the system decision to reflect the preferences of the end-user.

The essence of our work was to come up with an architecture that is flexible, extensible and capable of detecting and resolving conflicts. This was done by using an agent-architecture whereby the agent thinking and decision-making is controlled by policies. We also introduced the concept of the action suitability of a policy, which reflects how much an action proposed by a policy is suitable for handling an event. The action *suitability* is calculated dynamically based on '*fuzzy variables*', which reflects the user preference or the current state of the system. Fuzzy variables [Cox94 and Mot92] are variables that are neither 100% true nor 100% false. For example, a temperature of 80F may be considered as being rather hot, but it is also considered warm to some degree. To express this concept, we associate to the fuzzy variable *temperature* a value *hot* with a degree of truth (or degree of membership), since "*temperature is hot*" is partially true and

partially false at the same time. Our algorithm for conflict resolution is based on the *suitability* of a policy.

Section 2 of this paper will give more details on the architecture of our system In Section 3; we will provide more details on the policy and their representation. Section 4 describes our graphical notation for scenario representation. Section 5 and 6 demonstrates the decision-making mechanism through some scenarios, and finally, our conclusion will be presented in Section 7.

2. SYSTEM ARCHITECTURE

Our work is based on the negotiation agent approach [Gri94], and extends the work described in [Buh98], in which a multi-agents-architecture was proposed to resolve feature-interaction conflicts dynamically. The major shortcoming of the previous system is that it associated fixed priority levels to different actions. When a conflict occurs, the action with the highest priority was chosen. The main goal of our system was to provide a more flexible mean for resolving conflicts.

In our system, we use MediaPath [Mit97b] to control the calls. MediaPath is a multimedia communications system based on open standards, manufactured by Mitel Corporation.

Agents communicate with each other using MicMac [Mit97a]. MicMac is a *tuple space* [Gel85], which is an instance of blackboard-architecture. In such architecture, entities communicate by publishing events on the blackboard. On the other hand, agents that are waiting for events place a query on the blackboard, in which the agents expresses the parameters and constraints of the event they are waiting for.

Each physical device is represented by a ***device-agent***. The ***device-agent*** contains the policies responsible for handling and controlling the requests and actions of this device. The ***device-agent*** may contain multiple ***feature-agents***. A ***feature-agent*** contains all the policies that control the feature to which the device is subscribed.

The end user may also be represented by a ***user-agent***, which contains all the policies representing the constraints that the system imposes to this user, depending on the role of the user. The ***user-agent*** contains also the policies and the parameters that express the preferences of the user. When a user is assigned a certain device, the device loads the policies that are relevant to this device from the ***user-agent***.

3. THE POLICY REPRESENTATION

As explained in Section 2, each device is associated with a ***device-agent***, which is responsible for controlling this device. Each feature to which the device is subscribed is represented by a ***feature agent***, that is considered as being a part of the device agent.

Each user is also represented by a ***user-agent***, which contains all the policies that the system imposes on the user, as well as those that express the user preferences. When a device is being employed by a specific user, the ***user agent*** is contacted, and all the policies that are relevant to the device are loaded into the ***device-agent***. The ***device-agent*** also contacts the ***Policy Server*** to load the policies that are related to this device.

There exist two categories of policies, obligation and authorization [Mar97].

Obligations are policies that are associated with triggering events. When one of these triggering events occurs, the agent containing the policies associated with this event will have the duty of realizing the actions described in these policies. A positive obligation policy (**Op**) describes the action that the agent wants to perform when the triggering event occurs, while a negative obligation policy (**On**) describes the action that the agent seeks to prevent. Authorization policies are not associated with an event. They represent the permission in case of Positive Authorization (**Ap**) and the refusal of permission in case of Negative Authorization (**An**).

Two levels of registration exist, ***notification*** and ***handling***. An agent may want to be notified of the occurrence of an action to change its internal state or to start a set of actions. An example of this is a cost agent, which is responsible for calculating the cost of using a service, that wants to be notified when a call is established, in order to start calculating the resulting cost. The second level of registration is ***handling***. In this case, the agent wants to have a monopoly on the control of an event. Notification policies are processed before handling policies, as they are usually used for setting the context of the event.

Policies can also be associated with fuzzy-variables that reflects how applicable the policy is to the current context of the application. This is called the '***suitability***' of the policy. A policy that does not have associated fuzzy variables is interpreted as being 100% suitable.

When an event occurs, the device-agent will test the conditions associated with the policies waiting for this event. The device-agent will execute all the actions suggested by notification policies whose conditions are true. The handling policies will be examined next. Actions of the handling policies whose conditions are true will be suggested to the agent. The device-agent will choose one of the actions suggested by the policies of

the agent, or by other feature-agents contained in this device-agent. The choice of the agent will be based on the suitability associated with these actions. Note that *No-Action*, which means do-nothing, will always be suggested with a degree of suitability equal to zero. This is done to prevent actions with a negative degree of suitability from being carried out, which are the actions that received a denial of permission. More details on this mechanism will be provided in section 5 and 6.

```
<!ELEMENT Policy          (Event?, Subject+, Action, Target+, Condition*, Truth-Value?)>
<!ATTLIST Policy          id          ID                    #REQUIRED
                          type        (Op | On | Ap | An)   #REQUIRED >
<!ELEMENT Event           (Sender?, Receiver*, Parameter*)>
<!ATTLIST Event           id          ID                    #REQUIRED
                          event-class CDATA                 #REQUIRED>
<!ELEMENT Subject         (Entity-Name | Object)>
<!ELEMENT Action          (Action-Name , Parameter*)>
<!ELEMENT Target          (Entity-Name | Object)>
<!ELEMENT Condition       (Variable-Name, Operator, Value)>
<!ELEMENT Truth-Value     (Parameter-Name*)>
<!ELEMENT Sender          (Entity-Name | Object | any)>
<!ELEMENT Receiver        (Entity-Name | Object | any)>
<!ELEMENT any             EMPTY>
<!ELEMENT Entity-Name     (#PCDATA)>
<!-- Entity-Name is used to reference other objects -->
<!ELEMENT Action-Name     (#PCDATA)>
<!ELEMENT Variable-Name   (#PCDATA)>
<!ELEMENT Parameter-Name  (#PCDATA)>
<!ELEMENT Object          (Parameter)*>
<!ATTLIST Object          oid         ID                    #REQUIRED
                          name        CDATA                 #IMPLIED
                          symbol      CDATA                 #IMPLIED>
<!ELEMENT Operator        EMPTY>
<!ATTLIST Operator type   (lessThan | biggerThan | equals | notEqual |
                           biggerOrEqual | lessOrEqual) #REQUIRED >
<!ELEMENT Parameter       (Parameter-Name?, Value)>
<!ELEMENT Value           (String | Fuzzy-Value | Boolean | Numeric)>
<!ELEMENT String          (#PCDATA)>
<!ELEMENT Fuzzy-Value     ((Label, Degree-of-membership)*)>
<!ELEMENT Label           (#PCDATA)>
<!ELEMENT Degree-of-membership (#PCDATA)>
<!ELEMENT Boolean         (true | false)>
<!ELEMENT Numeric         (#PCDATA)>
```

Figure 1. Policies' Document Type Declaration.

Policies are represented in our system using XML. XML was developed as a standard by the World Wide Web Consortium as a meta-language, which is a language for describing other languages. It is a subset of the

Standard Generalized Markup Language (SGML). Many scientific communities have decided to adapt XML to define a standard for information exchange, like the Mathematical Markup Language [W3Cc] and the Synchronized Multimedia Integration Language (SMIL) [W3Cd]. The XML representation of a policy is given in **Figure 1**.

Using the XML grammar, the operator "," is used to denote a sequence of elements. The operator "?" means that this element can occur at the most, once, or not occur at all. The operator "+" denotes one or more occurrence of an element, while "*" denotes zero or more occurrences of an element.

In our system, policies may contain one or more objects, called subject. A subject is the entity responsible for executing the action of a policy. A policy may also contain one or more targets, where targets are the entities on which the action is performed. A policy may have an event associated with it, as in the case of Obligation Policies. Constraints are the pre-conditions that must be realized for the policy to be applicable. Constraints can place conditions on the subject, target or mode of the policy. It can also place a condition on the state of the system.

To increase the readability of the paper, we will use a notation inspired from the work of [Mar97]. The general format of this notation is given below, with optional arguments written within brackets.

Policy_ID type [registration-mode] [trigger] subject(s) {action} target(s) [when constraint(s)] [truth-value]

Figure 2. The policy notation.

4. THE GRAPHICAL REPRESENTATION OF A SCENARIO

There exist many techniques for scenario representation, such as Use Case Maps [Buh95 and Aym98]. However, these methods remain at the abstract level and are incapable of showing a specific scenario, or they are deficient in explaining the decision paths of the scenario. For this reason, we needed to develop a graphical notation to represent both the state of the agent system and the decision paths for each step in the scenario.

Let's start by explaining the symbols used in this notation. The rounded rectangle is used to represent an **Agent**. We put the agent name in the top part of the rounded rectangle, while the attributes that are relevant to the scenario are shown in the bottom section, as shown in **Figure 3**.

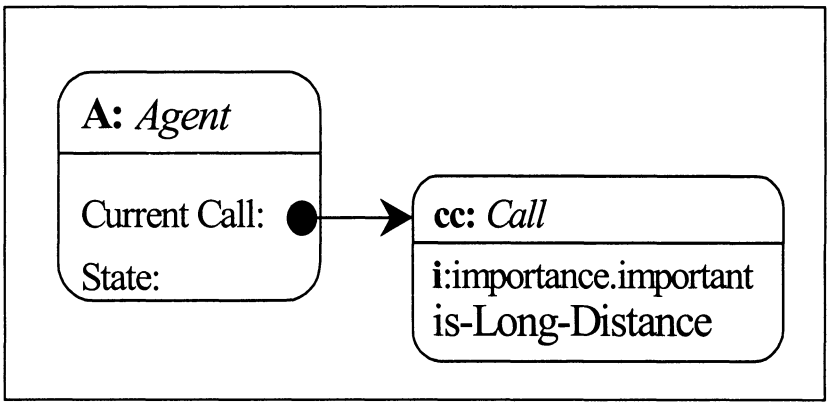

Figure 3. The Graphical Representation of an Agent.

A connector is used to represent an attribute that refers to another object. If no object exists, the connector will be grounded. Any agent, object or attribute can be represented by a symbol to facilitate referring to this element. For example, ***A.cc.ld*** refers to the parameter of Long Distance of the current call object of Agent **A**. To denote the specific label of a fuzzy variable, we append the variable name to the label name, separated by a dot. For example importance.urgent denotes the label "urgent" of the fuzzy variable "importance".

Actions are represented by ovals, within which we describe the action taken. An action may be associated with one or more groups of conditions. If any group of conditions is realized, the action will be proposed. A condition is symbolized by a hexagon and each group of conditions is connected to the action using the symbol of an encircled cross. The small circle bordering the conditions symbolizes that the action will be proposed if the condition is false. An action may have an associated truth-value that will be expressed by a down arrow callout, containing the parameters from which the truth-value is calculated. If no callout is shown, this means that the truth-value of this action is 100%.

The actions of notification policies are directly connected to the agent to symbolize that the agent will try to execute all the actions proposed. The actions of the handling policies are linked to the agent using an encircled dot to represent a decision point, as only one of these actions can be proposed. As explained previously, at each decision point we will have a ***No Action*** with an associated 0% truth-value.

If the Agent contains other sub-agents, this sub-agent will be connected to the Agent through an encircled dot to represent that the sub-agent can only suggest one action to the enclosing agent.

Events that are relevant to the scenario are shown in dashed rounded rectangles with arrows leading from the event to the conditions that depend on the event.

In the next section, we will use this graphical notation to represent a scenario of a conflict resolution between multiple feature-agents inside a single device-agent.

5. RESOLVING FEATURE INTERACTION INSIDE A SINGLE DEVICE-AGENT

In this example, we will explain how to use the agent architecture to resolve conflicts at the level of a single device agent. Let's suppose that we have a device agent containing three feature-agents, the Call Forwarding feature-agent (CF) and the Answer Call feature-agent (AC) and the Termination agent (T), which is responsible for accepting call requests. CW generates a call-waiting tone to alert the called party, whereas AC connects the calling party to an answering service. If **A** is already on the line when the second call comes in, should **A** receive a call-waiting tone or should the second call be directed to the answering service? The previous example is classified [Cam94] as a Single-User-Single-Component (SUSC), where the interactions occur because incompatible features are simultaneously in use by a single user in a single network component.

In this example, we assume that the user does not want to be interrupted during a long-distance telephone call. Thus, AC should take precedence over CW when the other call is long distance; otherwise, CW should take precedence over AC. The architecture of the agent is shown in **Figure 4**.

The device-agent contains the following notification policies.

- Pl_a1 **Op** [on *CR: CallRequest*] ***agent*** {set(0.6) } ***CR.importance*** [when not ***CR.isLongDistance***]
- Pl_a2 **Op** [on *CR: CallRequest*] ***agent*** {set(0.9) } ***CR.importance*** [when ***CR.isLongDistance***]
- Pl_a3 **Op** [on *CE: CallEstablishement*] ***agent*** {set(0.9* ***CE.importance***)} ***agent.currentCall.importance***

The policy *Pl_a1* says that when a local call-request is received, the agent should set the call importance (*C.importance*) to 60%. Rule *Pl_a2* says to set the importance of the call to 90% if the call is a long-distance call. Rule *Pl_a3* indicates that once the call has begun, the agent should reduce the importance of the call to 90% of its original value.

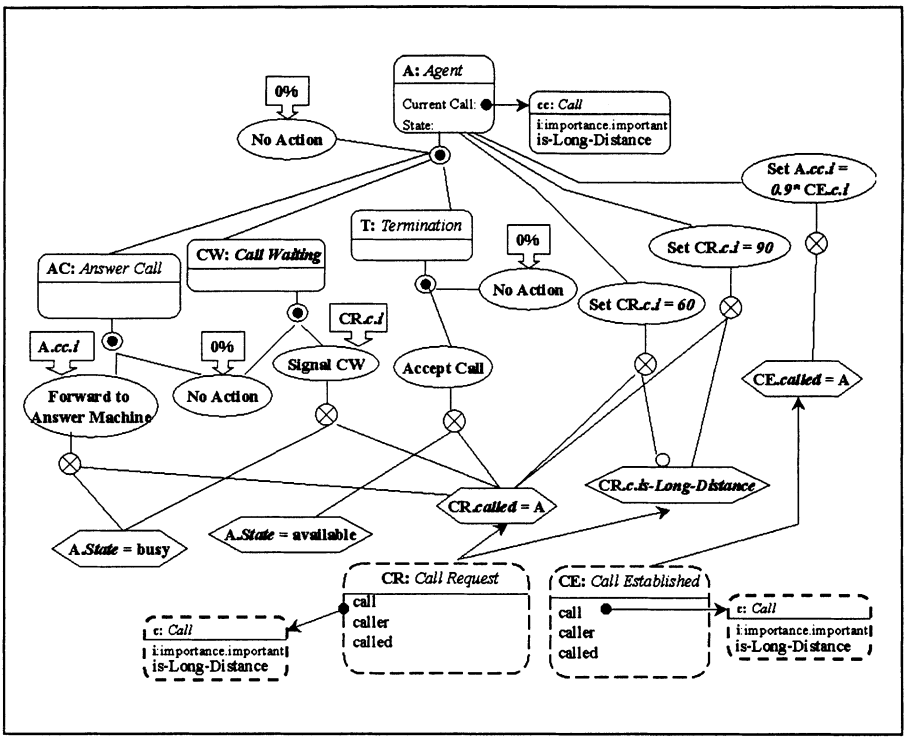

Figure 4. The Graphical Representation of the agent system.

The device agent contains three feature-agents, which are the Answer-Call agent, the Call-Waiting agent and the Termination agent.

The Answer-Call feature-agent contains the following policy.

- Pl_ac1 **Op** [on *CR: CallRequest*] ***agent.AC*** {forwardToAnswerMachine( ) } ***CR*** [when ***agent.state***= busy] [`Truth-Value` ***agent.currentCall.importance***]

This policy says that the call should be forwarded to the answer machine if the user is currently using the telephone. The truth-value of this rule will be equal to the fuzzy-parameter *importance* associated with the current call.

The Call-Waiting agent contains the following policy.

- Pl_cw1 **Op** [on *CR: CallRequest*] ***agent.CW*** {signalCallWaiting( ) } ***CR*** [when ***agent.state***= busy] [`Truth-Value` ***CR.importance***]

This policy says that the user should hear the call-waiting signal if the user is currently on the telephone. The suitability of this policy will be equal to the importance of the incoming call.

Let's suppose that a long-distance Call Request is sent to Agent A, while Agent A is available. From the Figure, we see that one action from the notification policies is true, which will set the importance of the Call Request to 90%. For the sub-agents Call Forward and Answer Call, only the

no action will be proposed, while the Termination Agent will have two actions proposed, that of Accept Call with 100% and No Action with 0%. The Termination Agent will choose the one with the higher truth-value, which is Accept Call, and will propose it to Agent A. Agent A will receive four proposals, three of them being No Action with 0% and the fourth being Accept Call with 100%. Thus, the Accept Call action will be executed. This step is shown in **Figure 5**.

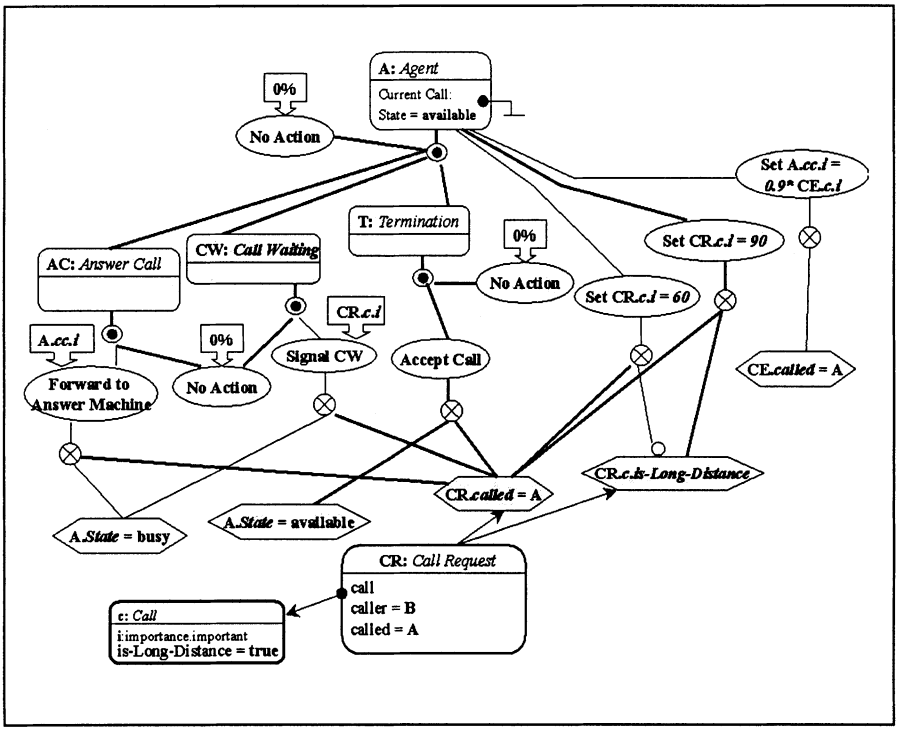

Figure 5. Step 1: Receiving a Long Distance Call Request.

Therefore, the call will be established and the Call Established event will be sent to Agent A, as shown in **Figure 6**. In this case, the state of Agent A will become busy, and the importance of the current call will be set to 90% of the importance of the call associated with the event, which is 90%. Thus, the importance of the current call will become 81%. On the other hand, all the sub-agents of Agent A will propose No Action with 0% truth-value, so no further action will be taken.

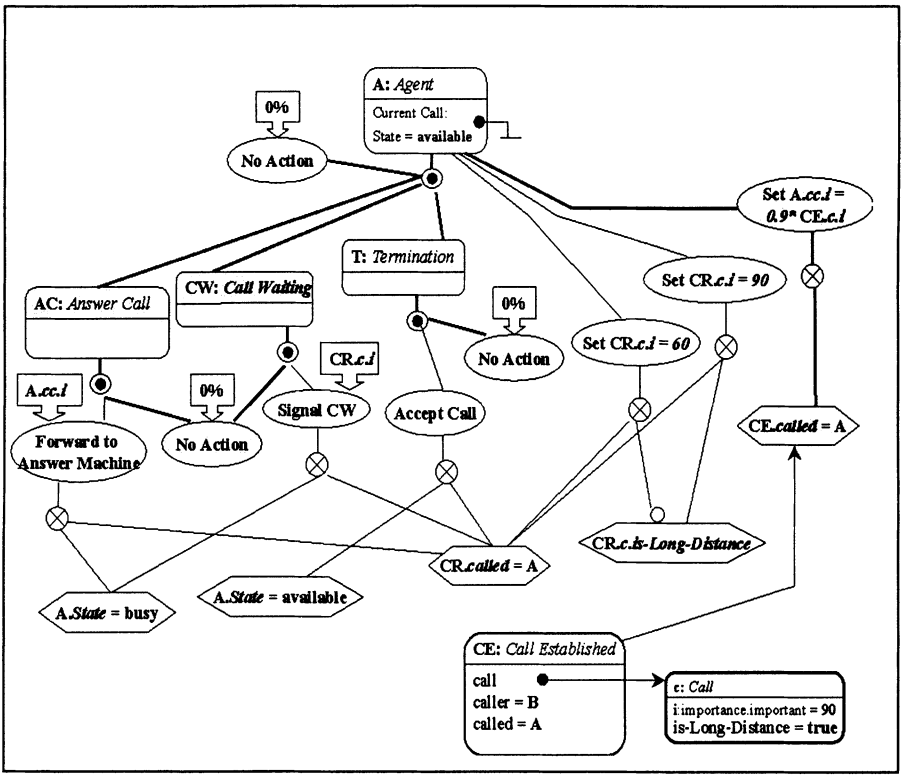

Figure 6. Step 2: Receiving a Call Establishment Event.

In **Figure 7**, we suppose that another long-distance call request was sent to the agent while the user was still busy. Similar to step one, only one notification policy will be triggered, which will set the importance of the Call Request to 90%. The Call Forward sub-agent will have to decide between No Action with a truth-value of 0% and Forward to Answer Machine action, whose truth-value is equal to the importance of the current call of agent A, which is 81%. Therefore, the Call Forwarding Agent will propose the latter action.

In the same way, the Answer Call will choose the Signal-Call-Waiting action, whose truth-value is equal to the importance of the current call, which is 90%. Agent A will have to choose between No Action with a truth value of 0%, Signal Call Waiting with a truth value of 90%, and Forward to Answering Machine with a truth value of 81%. So it will choose the action with the higher truth-value, this is the Signal Call Waiting.

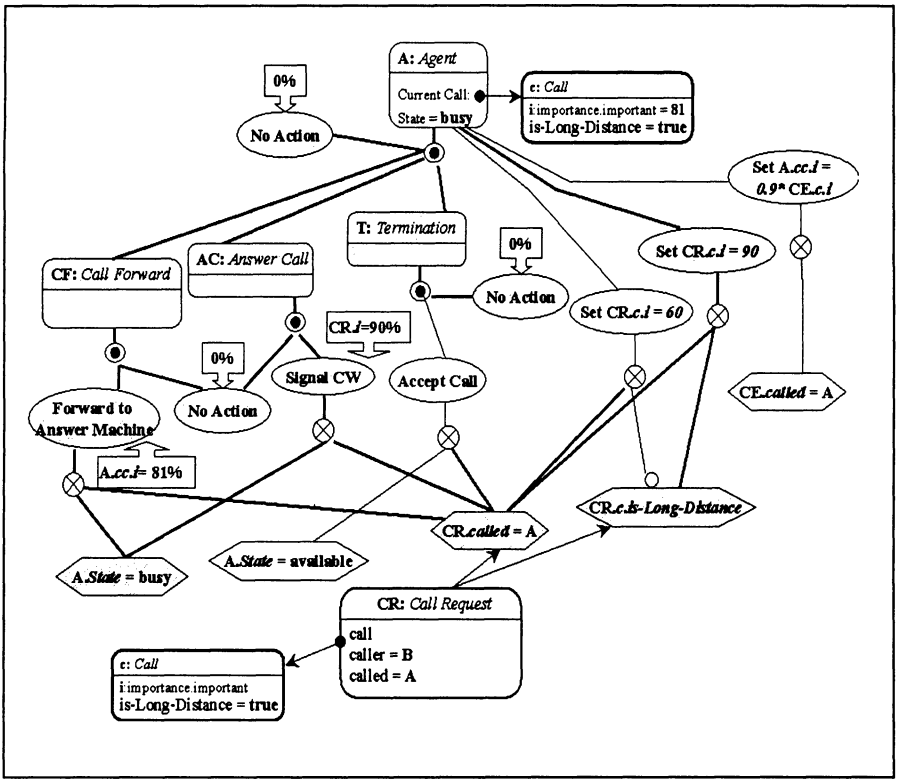

Figure 7. Step 3: Receiving a Second Long Distance Call Request.

In **Figure 8**, we suppose that the local telephone call was sent to Agent A while the user was still busy with a long distance telephone call. So only one notification policy will be triggered, which will set the importance of the Call Request to 60%. As in the previous case, the Call Forward Agent will propose the Forward to Answering Machine with a truth-value of 81%. The Answer Call Agent will propose a Signal Call Waiting with its truth-value equal to the importance of the Call Request, which in this case is 60%. In this case, Agent A will have to choose between No-Action with a truth value of 0%, Signal Call Waiting with a truth value of 60%, and Forward to Answering Machine with a truth value of 81%. The agent will choose the Forward to Answer Machine, as it is the action with the highest suitability.

As we've seen from this example, the graphical notations that we've developed help simplify the understanding of the system behavior, by allowing us to give a global view of each step of the scenario.

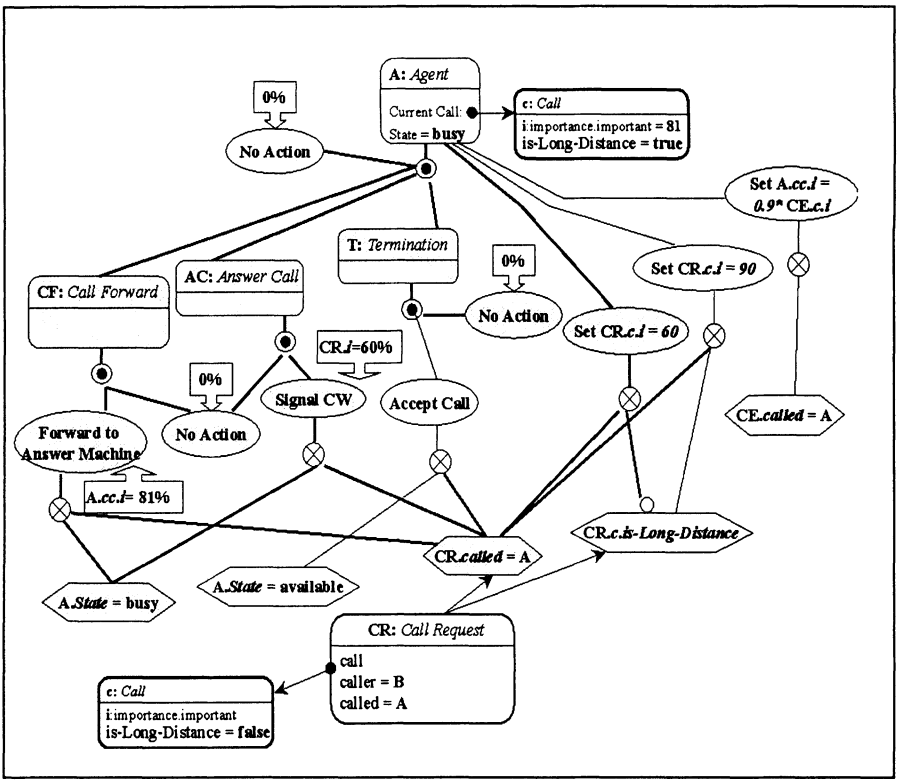

Figure 8 Step 4: Receiving a Local Call Request.

6. RESOLVING FEATURE INTERACTION BETWEEN MULTIPLE DEVICE-AGENTS

In this example, we will demonstrate how fuzzy logic can be used to resolve conflict between different device-agents.

Suppose that a user A wants to make a conference call with both B and C. A does not want any of the telephone calls to be forwarded to an answer machine or a secretary. Both B and C are not available. B has forwarded the telephone calls to the answer machine, while C has forwarded the telephone calls to a secretary. C has left a report with the secretary that should be read to the others in the conference call.

Let us take the first example of A sending a Call-Request (CR) to B.

The device-agent of A will post on the **blackboard** the following Call-Request.

- *CR.caller=A, called=B, id = X1*

The id of the call request is set by the originating agent to identify this request if further processing is required. For simplicity, let us suppose also that the only entity that is interested in this call request is the called agent B.

The device agent of B (B for short) contains two sub-agents, a Termination-agent (B.T) and an Answer-Call-agent (B.AC).

B will have an attribute *available* set to *false*. B also contains the fuzzy variable *forwardToVoiceMail* parameter set to 10%, indicating that it gives the possibility for the persons who call him to be forwarded to the answer machine, but without forcing them to.

B.T contains the following policies.

- *Pl_B_T1* ***Op*** *[on CR:CallRequest] B.T {acceptCall()} CR [when (B.available= true) AND (CR.called=b)]*

B.AC contains the following policy.

- *Pl_B_AC1* ***Op*** *[on CR:CallRequest] B.AC {forwardCall()} CR [when (B.available = false) AND (CR.called=B)] [Truth-value is (B.forwardToVoiceMail)]*

The output of B.T will be *no-Action*, while B.AC will choose to forward the call to the answer machine, with a truth value of 10%, which is the value of the *forward-To-Voice-Mail* parameter. The agent B will arbitrate between the output of the two modules and will choose the *forward-Call*. The result of this event will be a call forward on this call request.

The system will ask for the permission of the originator of this event, which is A.

A contains the following policy.

- *Pl_A1* ***An*** *A {forwardCall} CallRequest [Truth-value (A.noForward)]*

This rule indicates that A refuses to forward this call with a degree of truth equal to the value of the fuzzy input variable *no-Forward*. Let's suppose that A has set this parameter to 80%.

In this case, a No-Permission-Exception event will be generated, indicating that a suggested action has been refused. The **arbitrator agent** will be invoked to handle this event.

The **arbitrator agent** will recalculate the truth-value that was associated with the suggested action. The new value will be recalculated as equal to the original truth-value of the action multiplied by the weight that is accorded to the agent suggesting the action, minus the truth-value of the permission refusal action multiplied by the weight that is accorded to the agent refusing the action.

Suppose that we are using a passive arbitrator, which applies equal weights for every agent involved in the exception. In this case, the *forward-Call* truth-value will be evaluated to 10% - 80% = -70%.

B will re-evaluate the decision that it makes. Thus, B will have to choose between *No-Action* with 0% and *forward-Call* with -70%. B will choose the action with higher truth-value, so *No-Action* will be chosen. This means that the call will not be forwarded and the caller will receive a busy signal.

Now let us consider the case of A sending a Call-Request (CR) to C.

The device-agent of A will post on the **event-blackboard** the following Call-Request.

- *CR.caller = A, called = C, id = X2*

The device agent of C contains two sub-agents, a Termination-agent (C.T) and a Call-Forward-agent (C.CF).

C has the attribute *available* set to *false*. C contains also the fuzzy variable *forward-Call* set to 10%, to allow the calls to be forwarded to C's secretary.

C contains the following policy.

- *Pl_C1* ***Op*** *[on CR:CallRequest] C {set to 95%} forward-Call [when (CR.called=C) AND (CR.caller=A)]*

The previous policy is set by C to indicate that it is waiting for a call from A. When this call is received, the value of *forward-Call* is set to 95% to express the will of C to forward this particular call to C's secretary.

C.T and C.CF will contain the following policies.

- *Pl_C_T1* ***Op*** *[on CR:CallRequest] C.T {acceptCall} CR [when (C.available= true) and (CR.called=C)]*
- *Pl_C_CF1* ***Op*** *[on CR:CallRequest] C.CF {forwardCall( ) } CR [when (CR.called=C) AND (C.available = false)] [Truth-value (C.Forward-Call)]*

When the call request is received from A, C will set the value of *forward-Call* to 95%. The two sub-agents of C will suggest *no-Action* and *forward-Call* with a degree of truth of 95. Again, the system will ask for the permission of the originator of this event, which is A.

A will activate the rule *Pl_A1* and will refuse the call forward. A No-Permission-Exception event will be generated again and the **arbitrator agent** will be invoked again. The **arbitrator agent** will recalculate the truth-value associated with the suggested action. This value will be equal to 95% - 80% = 15%. Thus, the *forward-Call* action will still be the suggested action with the highest truth-value. The **arbitrator agent** will decide to override the objection of A and the call will be forwarded to C's secretary.

The examples shown in this section demonstrated the use of fuzzy policies. It explained how fuzzy logic constraints can be used to allow the user to alter the reaction of the system, depending on the user's own preferences, thereby giving the real control to the end user.

7. CONCLUSION

Telephony features may interact with each other in an undesirable or even an unacceptable way. Classical telephony systems resolve this problem by providing explicit instructions in their management software. With the increase in the number of features and the convergence of the worlds of data and voice, this approach is becoming less and less feasible.

In this paper, we proposed to use a multi-agents architecture to detect and resolve feature interactions. The end-user has the ability to alter the output of the interaction between the features by adding policies to the user-agent, thus making the system customizable.

ACKNOWLEDGEMENT

This work was partly funded by ***CITO***. It was the result of collaboration between ***the Multimedia and Mobile Agent Research Lab*** at the ***University of Ottawa***, the ***Network Computing*** group at ***National Research Council*** (NRC), and ***MITEL***.

REFERENCES

[1] Aho95 A. Aho, N. Griffeth, "Feature Interactions in the Global Information Infrastructure," in Foundations of Software Engineering, Washington, October 1995.

[2] Amy98 D. Amyot, N. Hart, L. Logrippo, "Formal Spe cification and Validation using a Scenario-Based Approach: The GPRS Group-Call Example". On-Line: http://www.sce.carleton.ca/ftp/pub/UseCaseMaps/wrroom98.pdf

[3] Bou94 L. G. Bouma, H. Velthuijsen, editors. "Feature Interactions in Telecommunications Systems", ISO Press, Amsterdam, 1994, 272 pp.

[4] Buh95 R. Buhr, " Use Case Maps: A New Model to Bridge the Gap Between Requirements and Detailed Design". On-Line: http://www.sce.carleton.ca/ftp/pub/ UseCaseMaps/oopslaUCwrkshp.pdf

[5] Buh98 R. Buhr, D. Amyot, M. Elammari, D. Quesnel, T. Gray, S. Mankovski, "Feature-Interaction Visualisation and Resolution in an Agent Environment", in [Kim98], p.135-149.

[6] Cam94 J. Cameron, N. Griffeth, Y. Lin, M. Nilson, W. Schnure, H. Velthuijsen, "A Feature Interaction Benchmark for IN and Beyond". In [Bou94], p. 1-23.

[7] Che95 K. E. Cheng, T. Ohta, editors. "Feature Interactions in Telecommunications III", ISO Press, Amsterdam, 1995, 223 pp.

[8] Cox94 E. Cox, "The Fuzzy Logic Systems Handbook", AP Professional, Cambridge, 1994, p.624

[9] Din97 P. Dini, R. Boutaba, L. Logrippo, editors. "Feature Interactions in Telecommunications Networks IV", ISO Press, Amsterdam, 1997, 373 pp.

[10] Fra96 S. Franklin, A. Graesser, "Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents". In Proceedings of the Third International Workshop on Agent

Theories, Architectures and Languages. Springer-Verlag, 1996. On-Line: http://www.msci.memphis.edu/~franklin /AgentProg.html
[11] Gel85 D. Gelernter , "Generative Communication in Linda". In ACM Transaction on Programming Languages and Systems, Vol 7, No 1, p80-112.
[12] Gri94 N. Griffeth, H. Velthuijsen, "The Negotiating Agents Approach to Runtime Feature Interaction Resolution". In Feature Interactions in Telecommunications Systems, IOS Press, Amsterdam, May 1994.
[13]Kim97 K. Kimbler, "Addressing the Interaction Problem at the Enterprise Level". In [Din97], p13-22.
[14]Kim98 K. Kimbler and L.G. Bouma, " Feature Interactions in Telecommunications and Software Systems V", IOS Press, Amsterdam, 1998, 374 pp.
[15]Mar97 D. Marriott, "Policy Service for Distributed Systems", Master's thesis, Imperial College of Science Technology and Medicine, London, UK, 1997, p. 131
[16]Mit97a Mitel Corporation Micmac, 1997. Http://micmac.mitel.com
[17]Mit97b Mitel Corporation Mediapath, 1997. http://www.mitel.com/MediaPath
[18]Mot92 Motorola Corporation, "Fuzzy Logic Education program", The centre for Emerging Computer Technologies, Motorola Inc., 1992
[18]Nwa96 H. Nwana , "Software Agents: An Overview" In Knowledge Engineering Review, Vol. 11, No 3, pp.1-40, Sept 1996. On-Line: http://www.cs.umbc.edu/agents/ introduction/ao/
[19]Oht94 T. Ohta, Y. Harada, "Classification, Detection and Resolution of Service Interactions in Telecommunication Services", in L. G. Bouma, H. Velthuijsen, [Bou94], p. 60-72.
[20]W3Ca World Wide Web Consortium, " Frequently Asked Questions about the Extensible Markup Language". On-Line: http://www.ucc.ie/xml/
[21]W3Cb World Wide Web Consortium, " Extensible Markup Language (XML)". On-Line: http://www.w3.org/TR/PR-xml.html
[22]W3Cc World Wide Web Consortium, " Mathematical Markup Language 1.01 Specification". On-Line: http://www.w3.org/TR/REC-MathML/toc.html
[23]W3Cd World Wide Web Consortium, " Synchronized Multimedia Integration Language (SMIL) 1.0 Specification". On-Line: http://www.w3.org/TR/REC-smil/
[24]Wal97 N. Walsh. "A Technical Introduction to XML".On-Line: http://www.arbortext.com/Think_Tank/XML_Resources/ A_Technical_Introduction_to_XM/a_technical_introduction_to_xm.html
[25]Wei97 M. Weiss, T. Gray. "Experiences with a Service Environment for Distributed Multimedia Applications". In [Din97].
[26]Woo95 M. Wooldridge, N. Jennings, "Agent Theories, Architectures and languages: a Survey". In: ECAI-94 Workshop on Agent Theories, architectures and Languages. Eds.(Wooldridge M., Jennings N.). Springer-Verlag, Berlin, 1995. pp.1-39.