USE OF MULTI-AGENT SYSTEMS FOR DETECTING FEATURES INTERACTION IN TELECOMMUNICATION SYSTEMS

MOHAMED ELAMMARI AND SARIA ELTALHI

Faculty of Information Technology, University of Garyounis, Benghazi, Libya
elammari@garyounis.edu  sasael114@garyounis.edu

ABSTRACT
A telecommunications system comprises multiple features and services, which are expanded in range and complexity by the addition of a new, advanced technology features. In order to avoid, detect and resolve potential feature interactions, many researchers have been trying to develop sophisticated methods, which frequently fail to anticipate the conflicts that occur when new modern features are added. The users of traditional methods in particular encounter many difficulties, especially regarding code management and the addition of new features to the system.

In this paper, a design stage approach for detecting feature interactions by using Multi-agent Systems MAS is presented. This approach is implemented in two stages. The first presents a model for describing visually (using UCMs notation) telephone features that help designers to obtain a general view or a global picture of the system. The second stage is agent specification where agent (derived from UCMs) represents the feature.

Keywords: Feature conflicts, Blackboard, Use Case Maps, Communications services, Detection.

1. INTRODUCTION
Changes in any software system may cause issues, such as undesirable interactions among features or components, which may occur in any stage. The literature defines this problem as the feature interaction problem.

In telecommunications systems, the feature interaction (FI) problem was recognized in the late 1980s and since then, many researchers in industry and academia have attempted to solve it, forming an active feature interaction community [1]. The series of international FI in telecommunications workshops have represented the results of this effort [2] [3] [4] [5].

“A feature interaction is any new behavior (desirable or undesirable), that results when combining two or more feature behaviors” [6].

The problem of FI can be explained easily to a human in a matter of minutes, but explaining it to a computer is a different matter [7].

Now, suppose that C subscribes to TCS and puts A in the screening list, and that B subscribes to CFB and sets the forwarding address to C. Now, if A dials B when B is busy, then CFB forwards the call to C. As a result, A can call C, which is a violation of the TCS’s feature and conflicts with the goal of TCS – hence, we say that features TCS and CFB interact. This scenario is shown in Figure 1.

Whilst this is a simple example, with an increasing number of features, the problem is typically much more complex.

Figure1. Interaction between TCS and CFB

2. EXAMPLE OF FEATURE INTERACTION:
Figure 1 illustrates an example of a problem that may occur between two common features. Some of the best-known services and their interactions are presented in [9].

Service TCS: Terminating Call Screening (TCS) allows a subscriber to screen incoming calls based on a screening list.

Service CFB: Call Forwarding on Busy (CFB) forwards incoming calls to another predetermined number when the subscriber is busy.

Service TCX (C has in the list)

TCS (C has A in the list)

Caller A
A dials B
B is busy
Forwarding to C
Answerer B subscribed to CFB
Answerer C subscribed to TCS

Figure1. Interaction between TCS and CFB

lead to undesirable behavior (negative effect); that is inconsistent with the user’s expectations, potentially leading to a breakdown of the system.

The FI remains a significant hindrance to the introduction of new features and to improving the telecommunications system [8].

The remainder of this paper is structured as follows: In Section 2, we present a simple example of FI problem. Section 3 provides the background to this work, in particular introducing features and feature interaction, and addressing currently recognized issues. In section 4 the review of the related work is presented. We propose the feature interaction detection method in section 5. This is followed by the summary of the study and the suggestions for further work.
3. BACKGROUND

3.1 FEATURE AND FEATURE INTERACTION

Pulvermüller [10] defines feature as “an extension to the basic functionality provided by a service,” whilst a service is explained as “a core set of functionality, such as the ability to establish a connection between two parties.”

“A feature interaction is understood to be any kind of unexpected interference among multiple features” [11]. Therefore, we can conclude that a feature interaction may occur when two features interact; these interactions are not necessarily a problem, as some can be desirable.

3.2 ADDRESSING FEATURE INTERACTION

Feature interaction problem can be handled using one of the following approaches [12] [13]: Avoidance, detection, and resolution

- **Avoidance:**
  The objective of an avoidance approach to prevent unwanted interaction. This approach is most suitable in the early phases of specification and design of features. It assumes that the cause of the interaction is known [14].

- **Detection:**
  The objective of a detection approach to determine whether or not a set of feature behaviors can cause conflicts due to unwanted interaction between them [15] [16].

- **Resolution:**
  The objective of a resolution approach is to find a suitable solution to undesirable interaction that may occur [17] [18]. Furthermore, categorization can subdivide the approaches into Off-line or On-line technique, as given in the introduction of [12].

  **Off-line** (Design time techniques) deals with problem before deployment.

  **On-line:** (Run time technique) deals with the problem after deployment.

3.3 FEATURE INTERACTION AND AGENTS

Agent technology has been receiving an increasing attention in the last few years and is quickly emerging as a powerful computing paradigm designed to deal with the complexity in dynamic distributed systems. The cause for the growing success of multi-agent technology in distributed systems is that the natural distribution allows for a natural decomposition of the system into several agents that act together to achieve a desired global goal [19].

The ability of agents to deal cooperatively with unexpected events in the environment is one of the most important aspects of their use in many applications: autonomy, pro-activeness, reactivity, collaboration and negotiation. In a distributed environment when interaction between features occurs unexpectedly, the agent is expected to find a solution [20].

4. RELATED WORK

Buhr and et al. [18] use the Use Case Maps (UCMs) notation to depict features in telecommunications systems. In this approach, tables generated from UCM behaviors provide a framework for humans to add information that will enable executable prototypes to be generated. Features are modeled as competing rule engines and interactions are detected and resolved by an agent system at run time by coordinating through a blackboard.

Khoumsi and Cherkaoui [21] proposed an agent-based method to demonstrate how feature interaction problem can be resolved by using agents. An interest of this approach is that, instead of modifying directly the interacting services, the authors use a static or mobile agent. In this approach every interaction is resolved without redesigning the two involved services and is based on a set of generic operations.

Amer et al. [22] proposed an agent-based architecture to detect and resolve feature interactions. The detection of conflict between features is determined at runtime using an Event Registration Protocol. The resolution of conflict is handled based on an adaptation of the fuzzy logic, the benefits from the flexibility and the semantic richness of policies and fuzzy logic to allow the end user to alter the system behavior, thus obtaining a more personalized service. In this approach, the reactions of an agent are determined through a set of policies. Agents may also contain fuzzy-variables whose values may be modified to alter the behavior of the agent depending on the user preferences. The user can also add policies to the agent, to obtain a more personalized service. When two agents propose to handle the same event using two different actions, the suitability of each proposed action will be calculated dynamically, based on the current system state and the denial of permission that these actions may have generated. The action associated with the higher suitability will be chosen.

Barbuceanu et al. [23] proposed architecture that can be applied to dynamically negotiate the provisioning of communication services based on the detection and resolution of feature interactions, they showed how simple and clear models of interaction and behavior can be combined in a generic negotiation architecture capable of automating the agreement reaching process between agents that, like in the global telecommunications system, have different preferences, policies and authority over resources and activities. At the basis of the architecture is an authority model of how agents can influence each other by setting obligations and interdictions upon their behavior.

5. FEATURE INTERACTION DETECTION METHOD

In the proposed method, we use feature descriptions from the user’s point of view. This
approach has two advantages; first, it can be applied at a very early stage of the design of new features; secondly, the descriptions can be quickly created, as the feature specification will be created using UCMs. Each UCM will represent a single feature and hold its specification. Combining the features and detecting possible interactions between them will be carried out automatically by Multi-agent system as described below.

The proposed action steps towards achieving the goals of the present study can be summarized in the following statements:

Given a set of feature requirements described using natural language:

1. Derive feature properties for each feature
2. Create UCM for each feature
3. From UCMs and feature properties derive an Accept Table for each feature
4. Derive Agent feature Specifications from UCM
5. Attempt to detect feature interaction by using agent and accept tables. The agent uses algorithm to detect conflicts

Detection of feature interaction can be determined by checking the accept table that contains all the possible conflicts of each feature; the agent uses the algorithm, illustrated in section 5.5.2 to detect conflicts between features.

5.1 DERIVING THE FEATURE PROPERTIES
The features are usually defined using natural language, which can lead to different understandings of a given feature; thus, how the feature properties are derived and represented are the key issues of a good feature interaction detection method. They also represent the biggest challenges, since the features are usually defined using natural language [24].

In this step, the feature properties are formalized by declarative transition rules, such rules consist of a Precondition, event, and a Postcondition formulated in a simple logic, as shown below [25].

Transition rules: Pre → Event Post

Automatic Recall (ARC) rule:
1. ARC (A) → Busy (A) → CallBackList(B,A)
2. ~Busy (A) → Connection (A, B)

where ARC (A) means that A is a subscriber to feature ARC, and Busy (A) indicates “user A is busy”. CallBackList (B,A) means that B is registered in list of A, and B is recalled when A becomes idle.

~ Busy (A) indicates “user A is not busy”, Connection (A, B) indicates “A is connected to B”.

5.2 CREATE UCM FOR EACH FEATURE
Figure 2 represents UCMs for the basic call model (UCM feature models) that contains both the caller’s and the callee’s scenarios in one model. Use Case Maps Notation (UCM) helps the designers to create a general view or a global picture of the system where each user has an agent responsible for managing subscribed telephony features, such as Originating Call Screening (OCS), where each user can communicate with its agent only, and agents can communicate with other agents. In UCMs, a stub is allowed to contain multiple submaps, which helps to describe dynamic situations through scenarios. These Use Case Map (UCM) scenarios are constructed from an informal collection of requirements. For more details on using UCM as a notation for describing features, see [26].

1) CNDB Feature
Call Number Delivery Blocking (CNDB): this service blocks the provision of the caller's number at the terminating side. Suppose that user x subscribes to the CNDB, when x dials y, x’s telephone number will not appear on y’s telephone display.

Agent Goal Model: this model represents the feature agent
Now, from UCMs we derive the Agent Model for each feature (this model will represent the feature agent) to
represent the Goal, Task, Precondition and Postcondition for each agent, where each agent represents a feature. Figure 4 shows the relationship between the UCM and the agent model. Table 1 and Table 2 show the resulting agent model for the Voice Mail and Terminating Call Screening Features.

![Figure 4: Derivation of Agent Model from UCMs.](image)

**Table 1: Agent Goal Model for VM feature.**

<table>
<thead>
<tr>
<th>Name agent: Agent VM</th>
<th>Goal</th>
<th>Plan</th>
<th>Task</th>
<th>Precondition</th>
<th>Postcondition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leave message</td>
<td>Record the call</td>
<td>Task:</td>
<td>precondition:</td>
<td>postcondition:</td>
<td></td>
</tr>
<tr>
<td>Record Call</td>
<td>Get(PIN origin), save Call</td>
<td>Start the record</td>
<td>Answer is busy</td>
<td>The call is record</td>
<td></td>
</tr>
<tr>
<td>Accept the call</td>
<td>Reject the call</td>
<td>Check the call's time</td>
<td>Busy</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 2: Agent Goal Model for TCS feature.**

<table>
<thead>
<tr>
<th>Name agent: Agent TCS</th>
<th>Goal</th>
<th>Plan</th>
<th>Task</th>
<th>Precondition</th>
<th>Postcondition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ref</td>
<td>CreateList(Get(PIN origin)), save Call</td>
<td>Task:</td>
<td>precondition:</td>
<td>postcondition:</td>
<td></td>
</tr>
<tr>
<td>Accept the call</td>
<td>Reject the call</td>
<td>if list is empty: then allow</td>
<td>not busy</td>
<td>Poc:Acc</td>
<td>Rej</td>
</tr>
</tbody>
</table>

5.3 BUILDING FEATURE ACCEPTANCE

To facilitate recognition of feature interaction conflict, an accept table for each feature should be defined (Table 3). The accept table is constructed from UCMs and feature properties based on the expected results of each feature. This table contains the following items:

- **Feature**: represents feature name.
- **Action/Condition**: demonstrates the parameters of the identified feature.
- **Accept**: demonstrates whether the calling is accepted or not, the result is based on the Action/Condition parameters.
- **Precondition**: this is the required condition to activate the feature. The precondition of each feature is captured from UCMs that represents the feature.

<table>
<thead>
<tr>
<th>feature</th>
<th>Action/Condition</th>
<th>Accept</th>
<th>Precondition</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCS</td>
<td>isInScreenList, caller, callee, x:Bool</td>
<td>Callee is not in List</td>
<td>User not busy</td>
</tr>
<tr>
<td>ARC</td>
<td>setCallBack, caller, callee, x:Bool</td>
<td>User is busy</td>
<td></td>
</tr>
</tbody>
</table>

**Table 3: Action Accept Table**

5.4 AGENT FEATURE SPECIFICATIONS

This section describes the components of agent feature specifications. In order to specify an agent feature, three components need to be specified: Agent, a Blackboard Architecture, and a Messaging system.

5.4.1 THE AGENT

The agent has specific responsibility, determined independently by the role that agent should play in the system. The agent is constructed of facts, rules and a specific goal to be achieved [27]. These agents are responsible for working collaboratively to detect the conflict and to connect telecommunication network users to each other taking into consideration the specific call features that users might have activated or deactivated. The agents use the algorithm illustrated in section 5.5.2 to detect conflicts between features.

5.4.2 MESSAGING

The messages allow direct communication between two agents. When the user agent (first agent) requests a connection with another user agent (if allowed), the second agent has to accept or reject the request. In case the connection is accepted, the first agent checks the action of the second agent to verify if there is a conflict with its action. In case of conflict, the sent message will be as follows:

- Massaged (From, to, "Sign of conflict")
- then the agent will end the connection.
- Messages can be sent to pass information back and forth between the two agents.
- Message (From, to, “Accept connection”)
- Message (From, to, “Reject connection”)

5.4.3 BLACKBOARD ARCHITECTURE

The blackboard is a classical abstraction that can be successfully applied to exchange of information in a distributed environment and in real world problem solving. The Blackboard architecture is widely used, due to its many benefits, as defined in [28] [29]:

- It supports the centralized system, implying a robust system with simple components.
- It consistently supports agents in cooperation in multi agent system, where agent uses a blackboard as a central repository to store and retrieve the data, bearing in mind that all the partial solutions made by other agents are proffered in cooperation.
- Flexibility of configuration: the knowledge sources (agents) can communicate and cooperate via common blackboard.
- Flexible problem solving: Blackboard architecture has been used in many applications; it is independent of any particular task, and it can integrate multiple problem solving modules.

The blackboard is used as a collection of agent information stored on it, which allows processing the current information. The information placed on the blackboard is public, available to all agents [30]. Since, in our approach, Blackboard is used as a central repository for all shared information and permits direct
communication between agents. This is in contrast to using point-to-point communication, which is contained two parts: the first part named as action blackboard, to be used for registering the action to be done by agents and the activated feature to be known to other agents (see section 5.1). The second part is named a tabular blackboard, to be used for registering the feature's acceptance table (see section 5.3).

5.5 DESIGN OF THE AGENT FEATURES INTERACTION
Multi agent systems deal with coordinating intelligent behavior among a collection of autonomous agents. Emphasis is placed on how these agents coordinate their knowledge, goals, and plans jointly in order to take action or to solve problem. To detect the conflicts between agents, some necessary procedures need to be taken into considerations, such as whether the agent is able to connect to another agent according to its subscribed features. The agent features store connection information between two users. The next subsections describe detection of feature interactions in more detail. Section 5.5.1 describes the actions performed by agent to detect the conflicts by taking the information on the caller and the callee, as well as the activated feature and activation condition, whereas section 5.5.2 illustrates the proposed algorithm detecting feature interaction.

5.5.1 ACTION AGENT
We define an "Action Agent" as having a Boolean value that identifies whether the action performed by agent to detect a violation of feature conflict occurred or not. When an action is generated, it is posted on the action Blackboard. Actions are tasks. To formalize these actions, a general form is prepared to confirm the ability to detect the conflicts. The action agent is comprised of:

Action-Agent (F_na, Caller_no, Called_no,?,Pre ,Bool)

Where,
F_na: feature name.
Caller_no: caller number.
Called_no: called number.
The special symbol ‘?’ means that the agent needs more information, and the agent will request this information from another agent.
The variable Bool returns true value if connection is allowed otherwise it returns false value.
The agent match both parameters in two actions to detect the conflict between two feature. The action accept table in section 3.2.3.3 can be used to return the necessary information and determined the Bool variable.

5.5.2 DETECTING FEATURES INTERACTION ALGORITHM
The algorithm described below aims to detect interactions among features. It is based on the action accept table and consists of necessary procedures that can be applied to find a conflict between agents when user agent activates a feature. First, the agent evaluates the preconditions, if preconditions are equal it indicates a conflict. Otherwise; the agent checks the actions against all agents' features; if it returns a true value, the agent detects a conflict.

The complete algorithm follows the steps outlined below:
1. Register the action of active feature on the blackboard.
2. Check if another action is on the blackboard; if so, go to step 3; else features interaction detection is finished (less than two features, so no interaction is detected)
3. Detect feature interaction by matching
   3.1 Precondition matching
      if Pre (F1) is equal to Pre(F2) then
      Message to (From,"To," sign to conflict")
      else
      3.2 Action matching
      3.2.1 Capture agent goals of two features
      3.2.2 Capture action/condition of two features based on action accept table and return the Bool value true (if accepted) or false (if refused)
      3.2.3 Compare the parameters in two actions.
      3.2.4 If two Action ( ) do conflict then message ("two features conflict")
      End if

6. APPLICATION
In this section, we apply the method to a telephone example in order to study the interaction between features and explain how agent can detect conflict between features. As an application of the proposed method, we will illustrate the interactions between OCS and CFBL. To detect whether a feature interacts with another feature, we need to know the goals of the features involved. Table 4 illustrates service names and their descriptions (Feature Goal).

<table>
<thead>
<tr>
<th>Feature shortcut</th>
<th>Feature Name</th>
<th>Feature description (Feature Goal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCS(A)</td>
<td>Terminating Call Screening</td>
<td>This service allows a subscriber to screen incoming calls based on a screening list</td>
</tr>
<tr>
<td>ARC</td>
<td>Automatic Recall</td>
<td>Automatically returning the last incoming call</td>
</tr>
<tr>
<td>OCS</td>
<td>Originating Call Screening</td>
<td>If a user tries to connect to a number that has found in list, then it blocks the call.</td>
</tr>
<tr>
<td>CW</td>
<td>Call Waiting</td>
<td>The Call Waiting service alerts the subscriber of an incoming connection when the subscriber is engaged</td>
</tr>
<tr>
<td>CNDN</td>
<td>Calling Number Delivery Blocking</td>
<td>This service blocks the provision of the caller's number at the terminating side</td>
</tr>
<tr>
<td>CFB</td>
<td>Call Forwarding on Busy</td>
<td>A subscriber of this service can forward incoming calls to another pre-determined number when the subscriber is busy</td>
</tr>
</tbody>
</table>

Table 4: Service names and their descriptions
6.1 FEATURES THAT MANAGE BUSY STATES
(PRECONDITION IS EQUAL)
There are many features managing busy states, such as Call Waiting (CW) or Call Forwarding on Busy Line (CFBL). Features A and B conflict if the precondition of feature A and precondition of feature B are equal to busy state. We use this example to illustrate the basic idea behind our approach.

6.1.1 EXAMPLE 1: CALL WAITING AND CALL FORWARD ON BUSY LINE

Detection of Interactions between CW and CFBL
Suppose that B subscribes to both CFB and CW, and that B is busy. At this time, if A dials B, then a nondeterministic behavior occurs: should the call be forwarded by CFBL or by CW?
The action of A will be register on blackboard

\[ \text{ActionCW (CW, A, B, User Busy, non) } \]

The agent will be detect the forward, and the agent of B will register the action on the blackboard.

\[ \text{ActionCFBL (CFBL, A, B, user busy, non) } \]

The agent will be matching and comparing actions and will find the conflict between two features. In this case, the feature interaction occurs when feature precondition conflict with that of another.

6.2 MANAGING FEATURES WHERE PRECONDITION IS NOT EQUAL
6.2.1 INTERACTION BETWEEN OCS AND CFBL
User A subscribes to OCS with C within his screening list. User B is a CFBL subscriber. When B is busy, A calls B and the call is forward to C. We applied the algorithm above to detect the interaction between Originating Call Screening and Call Forwarding busy line.
The agent feature of OCS will register the Action-A on Action blackboard:

\[ \text{ActionOcs (OCS, A, C, OCS is on, False) } \]

The user A is not allowed to connect to user C by OCS feature. Thus, false will be returned, based on the action accept table. The agent will detect the forward, whereby the feature CFBL allows the connection between A and C and the agent feature of B will register the action-B on Action blackboard:

\[ \text{ActionCFBL (CFBL, A, C, user busy, True) } \]

The agent will be matching and comparing two actions dependent on the algorithm above; thus the connection between user A and user B should not happen. Therefore, conflict between the two features is detected.

7. CONCLUSION
In this paper, we identify the problem of feature interaction and present a method that detects the conflict between features. The detect FI method has been described with the primary goal to derive the feature from the user requirements and present them through UCMs. The second goal is to represent each feature by an agent and detect the conflict between them by agent algorithm. The algorithm is used by agents to detect conflicts, enabling agents to cooperate by using a blackboard to store and retrieve information on potential conflicts. This algorithm is implemented in two stages; first, comparing the Precondition – equal implies conflict. Alternatively, if precondition is not equal, then comparing the parameters of action Agent, the conflict can be detect by knowing more information (precondition, who is the subscriber of the feature, screenlist, etc.) without knowing details of implementation. This method automatically detects the conflicts at design stage.

8. REFERENCES