FID: feature interaction detection tool

A. Sefidcon*, F. Khendek

Electrical and Computer Engineering department, Concordia University, 1455 De Maisonneuve W. Montreal, Que., Canada H3G 1M8

Abstract

Intelligent Network (IN) is a standard architecture developed by ITU-T and applicable to all telecommunications networks including PSTN, ISDN and PLMN. IN aims to ease the development and deployment of new telecommunications services. However, the problem of feature interaction hinders it from easily reaching this goal. In this paper, we discuss the feature interaction problem before introducing our tool for Feature Interaction Detection (FID). The FID tool takes as input features modeled with object-oriented templates. These features are described in terms of necessary resources and actions instead of detailed behaviors. The FID tool implements a new pragmatic detection technique and provides all the necessary components for using this technique as well as the necessary parts for its evolution in order to take into account new features and new causes of feature interactions. We validate our approach and tool against existing feature interaction benchmarks. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Intelligent network; Telecommunication services; Feature interactions; Detection; Pragmatic approach

1. Introduction

Intelligent Network (IN) is a service independent telecommunication network. Intelligence is taken out of the switch and placed in computer nodes that are distributed through out the network. IN provides a complete architecture and framework for the uniform creation, provision and management of advanced communication services.

A feature can be defined as an add-on functionality to the Plain Old Telephone Service (POTS). The valid operation of a feature is based on a number of assumptions about its necessary resources and execution environment. Features are usually developed in isolation of each other, probably at different times and by different people. When features are put together, they might compete for common resources and may change the environment and assumptions of each other. Thus, a feature can work well alone but act differently in the context of other features. In this case, the result will be an unexpected and/or undesirable behavior. This is known as feature interaction (FI) problem [5], which hinders IN from reaching its goal for rapid service development and deployment. Different examples of FI are presented in Refs. [2,4,5,7–11].

In this paper, we discuss IN and the FI problem. We introduce our tool (FID) for feature interaction detection and we discuss its applications. The remaining part of this paper is structured as follows. Section 2 gives a brief overview of major IN concepts. Section 3 discusses briefly the FI problem and the FI detection techniques. The FID tool is described in Section 4. In Section 5, we apply our tool to known feature interaction benchmarks and present some examples of newly detected feature interactions. The conclusion is given in Section 6.

2. Intelligent networks

IN is viewed as a framework to help the service providers in offering more advanced services. Intelligent Networks are modeled through the IN Conceptual Model (INCM). This model is published in the ITU-T Recommendation Q.1201. The call model is a representation of a sequence of procedures executed by an IN to set up, manage and clear an IN session between IN components. ITU-T has approved sets of capabilities for IN, called Capability Sets (CS). These CSs introduce a range of services and support the rapid customization of the services. Network providers take these standards as input and specify the details with the vendors for market-specific requirements.

The INCM has four planes called Service Plane (SP), Global Functional Plane (GFP), Distributed Functional Plane (DFP), and Physical Plane (PP). Each service in the SP of the INCM is a combination of some Service Features (SF). Each SF can be represented using several different types of Global Service Logic (GSL). There is one set of GSL per service feature and it
uses Service Independent Building blocks (SIB) in GFP. There is one set of Distributed Service Logic (DSL) per SIB and it uses Functional Entity Actions (FEA) and information flows in DFP. In PP the service logic programs may be installed into and executed by any physical entity. Service logic must be implemented in a service-independent way. This can be done by means of SIB. These SIBs are the programs written in programming languages and are presented in GFP. Services are composed of one or more service features. A service feature is the smallest part of a service.

3. Feature interactions

Feature interaction is understood as all kinds of unexpected interferences between several features (or instances of the same feature) that prevent at least one of them from performing its function(s) correctly.

As stated in the ITU-T recommendation for CS-1, the service interactions are described from the customer and user point of view. Service interaction applies to all interactions of the new services with the previous services.

During the last decade researchers have developed different approaches to tackle the FI problem [7–11]. These techniques can be classified mainly into four categories: detection, resolution, avoidance and management. A good survey and classification of these approaches is given in Ref. [2].

Detection consists of identifying the existence (presence) of interactions. Detecting interactions between two or more features involves an environment in which the actions of the features are observed to find out where the interaction occurs. For any method, the features must be described in terms of procedural, behavioral and/or working environment assumptions/characteristics. Applied detection algorithm can be varied according to the chosen methodology.

Detection techniques can be classified into two classes: runtime and off-line techniques. The runtime techniques consist of monitoring a real network for unexpected interactions between features. Analytical methods tackle the FI problem at the specification level of the model. They are also known as off-line techniques. An analytical method can be formal or pragmatic. Refs. [1,3] give an overall picture of the existing solutions for FI and their strengths and weaknesses.

4. Feature interactions detection tool

In this section, we introduce our tool, FID, for FI detection. This tool implements a pragmatic technique, which is based on the experience stemmed from the existing FIs. More precisely our approach is based on the known causes for FIs extracted from the Bellcore [5] and European [4] benchmarks. The necessary information to describe the features is derived from these causes and modeled with an object-oriented template. The detection technique is run on these feature descriptions and consists of three main steps: filtering, feature instantiation and detection. To describe a feature, all its associated components must be defined. The participants of the features and their triggering constraints and operations describe actions of the feature. Using filtering method all the possible interaction-prone scenarios are produced and the features are instantiated using the list of topologically different call scenarios. The detection algorithm, which consists of searching for all the causes of interaction, is run on these actual features to detect potential interactions and report the interactions between the features under consideration.

4.1. Causes of feature interactions

The causes of interactions can be classified into three main categories: assumption, resources, and operations. In each category there are different experienced causes of interaction. Ref. [12] gives examples from each category.

4.1.1. Assumptions

The correct work of each feature is based on some assumptions. These assumptions may include a particular call processing, a special billing system, or even how customers perceive the network operations. If one of the necessary assumptions of the feature is not satisfied, it simply cannot behave correctly.

4.1.2. Resources

Many difficulties with large distributed systems are also present in managing feature interactions. Each feature needs some resources in order to behave correctly. The activation of one feature can make the resources unavailable to another feature. This causes the feature to fail in its operation.

4.1.3. Operations

Operations of the features can be the causes of interactions in different ways. They can simply be contradictory (like read and protect-from-read), provide the race condition situation (when two features try to manipulate the same data) or destroy the assumption of each other.

To catch different causes of FI, we defined the necessary information for feature description and the necessary techniques to be applied on the description to detect the interaction.

4.2. Object oriented modeling of features

Our model for feature description is object-oriented. Various types of object models can be produced showing how object classes are related to each other, how objects are aggregated to form other objects, how objects use or inherit the services provided by the other objects, how one object is composed of some other objects and so on. For some classes of system, object models are the natural ways of reflecting the real-world entities that are manipulated by the system.
This is particularly true where the system is concerned with the processing of information about concrete entities such as subscriber, features and its participants [14]. Fig. 1 shows the model for categorising the necessary information items about a feature. In this model each box represents a class that describes one component. Informally, each (Subscriber) is a telephone user with the limitations assigned to his/her line. This subscriber subscribes for one or more (Features). Each feature has some limitation, special data, and an action. The formal participants (FP) describe feature action. Each FP is described by its originating (BCSM), terminating BCSM or both. In each BCSM the decision point (Trigger) in which this (Operation) happens is described.

4.3. FID tool

FID tool is a complete implementation of our approach. The previous explanations on features modeling and the detection process provided us with the requirements for the tool. To develop the tool, the object-oriented methodology was used. In analysis step, the requirements of the tool were analyzed to find the relevant components of the tool as well as the way to design the system. In the design step, the necessary objects and their relations were defined. Finally, in the implementation step, the system was implemented using an object-oriented language JAVA [13].

The FID tool consists of eight components: Editor, Choose-Features, Filtering, Detection, Report, View, Tool Evolution and Help.

4.3.1. Architecture of the tool

The architecture of the tool can be discussed based on its inputs, processes and outputs. However, for each system there are some important functional and non-functional requirements, which are considered in system development.

4.3.1.1. System input. This component allows for the user to enter feature specifications. Once the specification is entered, the user is able to modify and remove them whenever it is needed. The tool automatically keeps track of the relations between the features and their components and provides user-independent functions for updating them in case of modifications. Also, the tool compiles the input information and informs the user with possible error messages and asks the user to correct the information before saving them.

4.3.1.2. Filtering process. According to Ref. [6], it is just wasting the time and resources to process all the possible scenarios between a group of features to detect the potential interactions. To have an efficient method, the combination of filtering method with any other detection method is necessary.

The filtering idea can be simply explained in the following way: Assume, there are two features, one has \(n_1\) formal participants (e.g. Call Waiting has three Formal Participants, with ids 1, 2, 3) and the other one has \(n_2\) Formal Participants (e.g. Three-way calling has three FP with ids 4, 5, 6). There are \(n = n_1 + n_2\) formal participants. The number of possible scenarios is \(n^n\).

However, only topologically different combination has the meaningful semantics of the service application. The number of topologically different scenarios can be obtained by applying a formula and the list of topologically different scenarios can be produced using a recursive algorithm. However, not all items of this list are the combinations leading to interactions between two features. Let us consider the following features:

<table>
<thead>
<tr>
<th>Feature1</th>
<th>Feature2</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP1</td>
<td>FP5</td>
</tr>
<tr>
<td>FP2</td>
<td>FP4</td>
</tr>
<tr>
<td>FP3</td>
<td>FP3</td>
</tr>
</tbody>
</table>

Assume there is no relation between the formal participants, FP1 to FP5, so there is no interaction for sure. Applying the filtering conditions, the number of scenarios will be reduced. Therefore, using the total number of formal participants and applying the formula, the number of topologically different scenarios is derived, and the list is created. Then applying the filtering conditions reduces this list to be used for feature instantiation. The instantiation is the assignment of the actual participant identifiers to the formal participants. Assume two features one with three and the other with two FPs have been described. Assume further the filtering algorithm has been applied and the list of scenarios has been created. Feature instantiation means applying the items of the filtered list to derive the list of actual participants.
4.3.1.4. Database of feature descriptions. To provide better efficiency and performance, as soon as a feature and its associated components are entered to the system; it is saved in a database for further processing. Using this facility, the user does not have to enter the specification of a feature several times. The user is able to choose from a list of pre-entered features to detect the possible interactions between them. Furthermore, to give a fast result to the user, the tool provides the option of choosing between the detecting “only the first interaction” and “all the possible interactions” that can happen between the features under process.

4.3.1.5. Reusability. Based on the possible ways to detect the causes of FI, we defined sufficient information for feature description. The important question is “How to take into account new causes of feature interactions”. In other words, how to update the method as well as the FID tool without changing too much in the code when a new cause of interaction is discovered. To update the tool, a component that gets and integrates the new reference information was provided. The reference information is mainly about the “assumption”, “resources”, and “operations” as the causes of interactions.

4.3.2. Implementation decisions

At the implementation stage to support object-oriented programming, an object-oriented development approach must be followed. This means expressing the system requirements using an object model, designing using an object-oriented method and implementing the system using an object-oriented programming language such as C++ or JAVA.

Object models developed during requirements analysis may be used to represent both system data and its processing. They are useful for showing how entities in the system may be classified and composed of other entities. These objects should not include details of the individual objects in the system. This is a design consideration. Rather, they should model classes of objects representing real-world entities.

The aggregation relationship between objects is a static relationship. When implemented, objects that are parts of another object may be implemented as sub-objects. Their definition may be included in the definition of the object of which they are a part. For the language of implementation, we chose JAVA because of its special features in comparison to other programming languages [13]. It provides a security system to keep its programs well behaved. It provides built-in multi-threading and the programs can be designed to do several things at once. JAVA has the built-in internetworking capabilities and allows means of accessing multiple computers and distributed applications. It also has its own garbage collection features and the programmer does not have to take care of memory pointers and memory leaks. Because of those advantages we chose Java as the programming language to implement our tool.

For example:

<table>
<thead>
<tr>
<th></th>
<th>FP1</th>
<th>FP2</th>
<th>FP3</th>
<th>FP4</th>
<th>FP5</th>
</tr>
</thead>
<tbody>
<tr>
<td>12111</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>12233</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

In the first case, there are only two actual participants. Actual participant with id 1 has to carry out the roles of FP1, FP3, FP4 and FP5. Actual participant with id 2 has to carry out the role of FP2.

The FID tool provides the facility to apply the filtering algorithm on the feature specifications. The result of the filtering algorithm is saved in a data structure to be used by the detection algorithm.

4.3.1.3. Detection process. The main goal of the tool is to detect interactions between the specified features. The detection algorithm consists mainly of necessary procedures to verify the presence of the causes of interactions. Controlling the values of common parameters of the actual participants of the features in different situations detects the interactions. The complete algorithm is shown in Fig. 2.

Each procedure applies one rule to find a specific cause of interaction. The FID tool provides a component, which is the implementation of the detection algorithm. This component uses the result of the filtering algorithm to instantiate the features and provides the list of actual participants. Then, the detection algorithm uses the list of actual participants to apply the conditions and detect the interactions between the features.

4.3.1.4. Database of feature descriptions. To provide better efficiency and performance, as soon as a feature and its

---

(1) Check the (Naming Assumptions)
(2) Check the (Supporting Networks)
(3) Check the (Administrative Domains)
(4) Instantiate the Features and FOR (all the actual participants of all the features under process)
(5) IF (More than one feature are activated in this FP)
(6) IF (Similar Detection Point)
(7) IF (Same Activating Priority)
(8) IF (Similar Operation)
(9) Check Resource Availability (CPE-Signal)
(10) Check Resource Availability (Query/Call)
(11) Check Data Availability
(12) ELSE (Different operation)
(13) Check the Operation Compatibility
(14) Check if one operation is Call Control
(15) Check if one is changing an object while other is using it
(16) Check If there is non-atomic operation
(17) ELSE (Different Activating Priority)
(18) Check the Result of one operation on the Criteria of the other
(19) ELSE (Different Detection Point)
(20) Check the Result of one operation on the Criteria of the other
(21) ELSE (Maximum one feature is activated in this FP)
(22) Check the Result of one operation on the Criteria of the other

Fig. 2. The complete detection algorithm.
4.3.3. Tool evolution

As mentioned previously, the causes of interactions, which are classified into assumptions, resources, and feature operations are general and can be extended to cover new technologies and IN capability sets. For example, new assumptions may be required for the correct behavior of the features, the features may need new resources and new operations may be defined. The tool provides the possibility of updating reference information using provided data by the user, based on the new technologies. Therefore, the tool can be updated according to the new features and new causes of FIs. To add new operations to the reference information, the user of the FID tool can simply use the “EXTEND” component of the tool and enter the new contradictory or non-contradictory operations.

Having entered the new contradictory operations, the reference information is updated. This information is used for editing data in Detection Point form for describing the operations of the feature and also for detection the interactions between the features based on the “operation compatibility” condition.

4.3.4. FID components

Each component provides a facility to the user. A user can enter new features, new information, choose from already existing features, filter the interaction-prone call scenarios between features using the Filtering component and detect the interactions by running the detection component. Fig. 3 shows the user interface of the tool.

The Editor provides the complete environment to add, modify and remove Subscriber and Feature specification. It saves all this information permanently in the tool databases. It keeps track of the relation between features and their subscribers. The user can introduce new Subscriber followed by his/her associated features or enter the specification of a new feature for already defined subscriber. The user always has the possibility of modifying the specification and updating the database. A list of saved features associated with their subscribers allows the user to choose the features for further processing.

After selecting features from a list, the user can run the filtering algorithm in order to create the list of interaction-prone call scenarios. Application of the filtering method is done on several steps. Each step corresponds to one condition that must be satisfied if the combination is to be interaction-prone. After applying each condition the number of items in the list is reduced and is shown in messaging window.

The detection algorithm can be applied to the items of the reduced list. The result of this will be the report of interactions between the selected features and the causes of these interactions, which can be used for further process such as resolution. All the interaction of the tool and the user is provided through the messaging window.

In most cases, it is necessary to submit the processing results for further study or resolution of interaction. The processing results are saved in a printable file. The tool provides this facility by creating a report. The result of the filtering and the detection programs will be saved into this printable file.

The Help component provides guidelines for the user. It helps the user in walking through the components of the tool. It helps the user to understand the correct order of running the tool components. Beside the user manual it provides the interactive prompt to keep track of the order for applying the programs and help the user understand the mistakes.

5. Application

To validate the FID tool, we checked it against the known feature interactions. The application of our tool to the features introduced in the benchmarks shows that the tool detects all the interactions. It is important to know that the experienced feature interactions are not the only possible ones. There are maybe a lot more inexperienced feature interactions. To measure the ability of our tool in detecting...
new feature interactions, we randomly chose different combinations of features to see whether there are interactions between them or not. Some of the newly detected feature interactions are shown in Table 2.

Some of the cases are common to the Bellcore benchmark (BB) and the European benchmark (EB). In Table 1 the abbreviations used for the features are presented and in Table 2 some examples of applying our detection technique for the combination of these services are presented.

### 6. Conclusion

Along with more and more emerging IN-service platforms for telecommunication services in fixed and mobile networks, feature interactions between IN services need to be handled. This can be achieved through an efficient and systematic approach for feature interaction detection. Our FID tool for FI detection is based on a pragmatic approach, which is based on the observed cases and causes. Despite the fact that we cannot anticipate on the coming features and their potential interactions, the proposed approach/tool has the following advantages:

- It has the simplicity of the pragmatic methods.
- It has the accuracy of the formal method, because it considers all the possible scenarios, which can happen between two features.
- Feature description is simple and does not need detailed design information about the features.
- Modeling template embraces all feature components. Since the model is based on the object-oriented methodology, it can evolve easily.
- The approach covers all feature interactions introduced in the Bellcore and European benchmarks.
- It detects new feature interactions, which have not been experienced yet.
- Using the filtering idea, it avoids state explosion, because it reduces the number of scenarios by applying different conditions.
- The FID tool is user friendly.
- As the most important advantage, the FID tool can be extended easily and can take into account new features and new causes of interaction without big changes to the program.

### Acknowledgements

This work is partially supported by NSERC Grant # OGP0194234 and FCAR Grant # 98-NC-1821.

### References


Azimeh Sefidcon received an Engineering degree in Electrical Engineering from Shahid Beheshti University, Tehran, Iran, in 1992 and the MASc degree in Electrical Engineering with the emphasize on telecommunication networks from Concordia University, Montreal, Canada, in 1999. She worked for five years in industry with concentration on software engineering, before returning to Concordia University in 1997 for Graduate studies. She is currently working as a telecommunication engineer at Ericsson Research Canada. She is also a PhD candidate at Concordia University. Her areas of interest include telecommunication services, Intelligent Network, and Mobile network.

Ferhat Khendek received an Engineering degree in Computer Engineering, option Software, from the University of Tizi-Ouzou, Algeria, in 1986, and the MSc and PhD degrees in Computer Science from the University of Montreal in 1989 and 1995, respectively. During his PhD studies, he held a research fellowship from the IBM Center for Advanced Studies in Toronto. Ferhat Khendek is currently an associate professor with the Electrical and Computer Engineering Department at Concordia University. His current research interests are mainly in the use of formal methods (SDL, MSC, UML) for the design and analysis of communication software and service engineering techniques. Ferhat Khendek is a member of l’Ordre des Ingénieurs du Québec (OIQ), ACM, IEEE and the SDL Forum Society.