A Tool for the Identification of Interaction-Prone Call Scenarios

Dirk O. Keck
Institute of Communication Networks and Computer Engineering (IND)
Prof. Dr.-Ing. Dr. h. c. mult. Paul J. Kuehn
University of Stuttgart
Pfaffenwaldring 47, D-70569 Stuttgart (Germany)
E-mail: keck@ind.uni-stuttgart.de

Abstract. Service interactions are an obstacle to the goal of today's network technologies to introduce new services rapidly, cost efficiently and with high quality. In a contribution to the 1997 Feature Interaction Workshop, Kimbler [1] has presented an approach for handling service interactions in the service life cycle, which introduces the concept of *Interaction Filtering,* i. e., the reduction of the number of problematic service combinations, as a first step towards interaction detection.

In this contribution, a prototypical tool is presented that carries out Interaction Filtering of IN services. It is based on the notion of a call scenario. A call scenario is the association of a number of telephone users that are tied together by one or more services they use in a communication relationship. For each combination of services, a sometimes large list of such call scenarios can result. Such a list is created automatically by the tool described here starting from a simple description of each service consisting of the standard application case of this service (the so-called service scenario), and the triggers and detection points used by this service. The length of this list of scenarios is reduced by topological criteria (conditions for the reachability of service triggers, symmetry relationships) and by other criteria considering the usage of data and resources common to the services. The application of these relatively simple criteria leads to a significant and transparent reduction of the number of interaction prone call scenarios and increases the manageability of the problem's complexity.

The list with the reduced number of interaction-prone call scenarios may serve as an input to a further investigation, using, e. g., formal techniques, manual analysis or testing.

1 Introduction

The Intelligent Network (IN) is an architectural concept that has been introduced in order to support the rapid and cost efficient introduction of new services into a telecommunications network. It is based on the concept of separation of the basic switching functionality of the network from the service processing capabilities, and on the concentration of the latter capabilities into a comparatively small number of nodes in the network.

A major obstacle to the introduction of new services is the service interaction problem. A service interaction is the phenomenon that two or more services that are activated in combination behave differently from the expected behaviour of each individual service. This malfunction can lead to the annoyance of customers, or even require significant modification or the withdrawal of the conflicting services, which clearly contradicts the IN's goal of rapid, cost-efficient and independent service development.

Service interactions emerge from three different levels in the IN system. On the *logical level*, interactions manifest itself by inconsistencies or contradictions in the requirements and aims of a service. On the *abstract architectural level*, interactions are caused by restrictions in the capability of the underlying architecture, i. e., in this contribution, the IN infrastructure, and in incomplete information of the services on the other services involved in a session. Finally, on the *concrete architectural level*, all the physical and implementation specific restrictions of real systems can be the source of additional problems.

Usually, the first measure taken to tackle the service interaction problem is the detection of interaction. To achieve this, a number of methods usually are applied, like, e. g., formal description and verification techniques [2], or systematic testing. All of these techniques require a significant amount of work and thus are relatively expensive. Furthermore, the number of possible service combinations that are potential sources of interactions is very large and grows almost exponentially with the number of services present. It would be helpful to have a method to screen out the irrelevant cases of service combinations in order to be able to concentrate the effort onto the analysis of the dangerous, interaction-prone scenarios. This task is carried out by the prototypical software tool for identification of interaction-prone service scenarios presented in this contribution.

The rest of this paper is structured as follows: In section 2, the filtering approach underlying the tool is described in detail. In section 3, the design principles and the tool architecture are presented, and section 4 contains case studies that illustrate the application of the tool. In section 5, some other approaches and their relations to the approach presented here are discussed, and section 6 concludes this contribution.

2 The filtering approach

2.1 The concept of filtering

The service interaction problem is characterised by an enormous number of possible service combinations that have to be checked for the presence or absence of interactions. This checking usually is a complicated and time consuming task, irrespective of the methods applied.

It is obvious, however, that there exist relatively simple indications that a certain service combination is likely to exhibit interactions or not. Using these indications, a quick and inexpensive pre-selection of interaction-prone constellations can be carried out, supporting a selective analysis of the interesting scenarios and avoiding the waste of resources for non-critical cases. Thus, the filtering concept (Kimbler [1]) strongly supports the IN goal of a fast and cost-efficient introduction of new service.

2.2 Scenarios and topological conditions

2.2.1 Service scenario

First, some basic IN concepts and the term of the *service scenario* used throughout this paper have to be introduced. On the Distributed Functional Plane (DFP) [3] of the Intelligent Network Conceptual Model (INCM), call processing is modelled using so-called *Basic Call State Models (BCSM).* These BCSMs are a sort of extended finite state machines modelling the different phases that are reached during call processing.

Within these BCSMs, there are well-defined Detection Points (DP). At these DPs, it is possible to interrupt the normal call processing and send a request or a notification to a service logic, which in turn is able to react on requests and instruct the call processing to proceed in a certain manner.

Each IN service is characterised by its service logic. This service logic initially is activated by a trigger, the so-called *Initial Detection Point*. This trigger is part of a transition between two states of a BCSM and marks a special condition which is met at a certain state (or *Point In Call*) during call processing. After activation, the service logic has several possibilities to influence call processing, like, e. g. continuation with no changes, release of the call, connection to another party. Furthermore, it is able to request the activation of further DPs, so-called *Event Detection Points (EDPs)*. These EDPs differ from the TDP as they do not trigger a new service, but inform an already active service on a certain event. There is a strict priorisation among the different kinds of DPs; notifications take priority over requests, EDPs over TDPs. For more details, the reader is referred to the IN standards or introductory books [4, 5].

The BCSMs usually are associated in pairs of Originating and Terminating BCSMs. Furthermore, as already mentioned, an IN service may initiate the creation of additional, concatenated basic call state models. This results in a typical setup for a specific IN service, that usually consists of a number of BCSMs involved directly or indirectly in the processing of the IN service, a number of participants, so-called *formal participants*[1] these BCSMs are assigned to, and the Initial Detection Point (trigger) and further Event Detection Points used by this service. These constituents make up a *service scenario,* which is one of the fundamental concepts used in this contribution.

This term is clarified by the examples depicted in Figure 1. In (a), the service scenario for a simple call with a Terminating Call Screening (TCS) service is displayed. It consists of an O-BCSM belonging to the formal participant A, the caller, and a T-BCSM belonging to B, the called party with active TCS. The presence of a service's initial trigger is symbolised by the grey shading of the triggering BCSM.

In (b), a service scenario for the Call Forwarding Unconditional (CFU) service is shown. In CFU, a call that reaches the party subscribing to CFU is immediately forwarded to a third party, the forwarding destination. The service scenario of CFU consists of an O-BCSM and a T-BCSM of the first leg of the call, the initial call, and another O-/T-BCSM pair of the second leg, the forwarded leg. It has got three formal participants, the caller, the forwarding party, and the forwarding destination. The CFU service is triggered at the T-BCSM of the first leg and initiates the second leg.

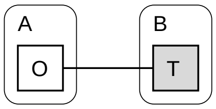
2.2.2 Service and call scenarios

A service scenario is an abstract description form of an IN service. It can be transformed to a concrete *call scenario* (or *call configuration*), if *actual participants*[2] are assigned to the service scenarios' formal participants. With respect to the service interaction problem, this transformation only makes sense, if at least two service scenarios (of two different services or of the same service) are transformed into a common call scenario. In the case study presented in section 4.2, this assignment is described and visualised in Fig. 3. The assignment of actual participants to the call scenarios' formal participants can take place in many different ways resulting in a combinatorial problem. This

[1] In this contribution, capital characters are used in order to label *formal participants.*

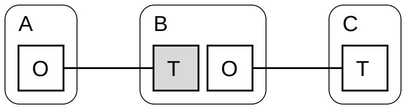
[2] In this contribution, Arabic numbers are used in order to label *actual participants.*

(a) Simple association of O- and T-BCSM



Example: *Terminating Call Screening*
A = Caller
B = Party with active TCS

(b) Service scenario with coupled T- and O-BCSMs



Example: *Call Forwarding Unconditional*
A = Caller
B = Forwarding party
C = Forwarding destination

Figure 1: Examples of service scenarios with formal participants. The locations of the initial detection points of the example service features are indicated by the shaded BCSMs.

problem has been studied in an earlier contribution [6], the major results of which are repeated in the following sections for the sake of completeness.

Let there be two service scenarios S_1 and S_2 with n_1 and n_2 formal participants respectively (S_1 and S_2 may denote different service scenarios as well as two instances of the same scenario). If these $n = n_1 + n_2$ formal participants are assigned to up to n actual participants, a theoretical number of n^n possible call scenarios involving S_1 and S_2 result. However, only topologically different assignments are relevant. Each of the actual participants carries the role of at least one formal participant and thus can be represented as a set of formal participants. We now have the problem of mapping n formal participants to up to n non-distinguishable disjoint sets, each of which representing one actual participant. The number of these mappings k_n can be calculated using a combinatorial formula for putting n different things into m like cells, with $m = 1, \ldots, n$ [7]. The non-recursive form of this formula is

$$k_n = \sum_{m=1}^{n} S(n,m) \tag{1}$$

where $S(n,m)$ are the Stirling numbers of the second kind,

$$S(n,m) = \frac{1}{m!}\left(\sum_{i=0}^{m}(-1)^i \binom{m}{i}(m-i)^n\right). \tag{2}$$

This relation also can be expressed more elegantly using the recursive formula [8]

$$k_n = \begin{cases} \displaystyle\sum_{i=0}^{n-1} \binom{n-1}{i} k_{n-1-i} & \text{for} \quad n \geq 1, \\ 1 & \text{for} \quad n = 0. \end{cases} \tag{3}$$

If S_1 and S_2 are instances of the same service, the symmetry between the formal participants of S_1 and S_2 additionally is exploited, leading to a further reduction of the number of call scenarios. The resulting list of call scenarios serves as a base for the application of the different filters of the subsequent filtering steps.

2.2.3 Basic topological condition

In [6], it has been shown that there exists a simple necessary condition for the presence of interactions between two services in a specific call scenario, that can be obtained using the information where the services' triggers are located in these scenarios:

One pair of associated BCSMs containing an initial trigger detection point of one service must be common to both service scenarios: It is only necessary to study interaction scenarios where the association of the O- and T-BCSM containing the initial trigger detection point of one service is also contained in the second service scenario.

This condition is applied to the set of call scenarios created by the combinatorial scenario generation described in the previous section. It leads to a significant reduction of the number of scenarios that has to be regarded in subsequent analysis steps (see section 4.1). A generalisation of this condition to combinations of more than two services should be manageable, but probably is difficult to prove or to validate because of the exploding number of topologically different call scenarios.

2.2.4 Further topological conditions

Practical experience has shown that not all scenarios left over after application of the above basic topological condition are equally relevant. Especially topologies where two or more formal participants of one service instance are mapped to the same actual participants very often lead to problematic or irrelevant call scenarios. Such scenarios are marked by this filtering step, dividing the set of scenarios left over by the above condition in a set of non-problematic call scenarios, where all formal participants of each service scenario have been mapped to different actual participants, and in a set of problematic call scenarios, where at least two formal participants of one service scenario have been mapped to the same actual participant in the call scenario.

In a later version of the tool, it is planned to be able to specify additional constraints that explicitly screen out scenarios that are not permitted by the service implementations, like, e. g., configuring a call forwarding to oneself. In the current version, these scenarios (which usually fall into the set of problematic scenarios described above) are sorted out manually.

2.3 Non-topological criteria

In this section, the non-topological criteria are discussed, by which interaction-prone scenarios are selected from a list of call scenarios which has been established exploiting the topological conditions discussed above. The criteria described in this direction support the identification of scenarios which are interaction-prone on the *abstract architectural level*, i. e., the interaction is caused by the way a service is realized in the IN environment, using trigger detection points, information elements, resources, etc. Interactions on the *logical level* are not covered currently, as far as they do not manifest themselves also on the abstract architectural level. However, it should be possible to identify additional filtering criteria that cover – at least partially – also that level.

2.3.1 Trigger collision

An important, but relatively simple source of service interaction can be the triggering structure of the services involved. Here, especially a *trigger collision* is a potentially interaction-prone situation. A trigger collision is a situation where the two services are triggered by the same detection point at the same participant. In reality, this leads to ambiguous situations that have to be resolved by mutual exclusion, disjoint conditions or assignment of priorities. These situations are identified and marked by the tool.

2.3.2 Data conflicts

A service has the ability to act on call instance data (CID) like, e. g., the Calling Number, the Dialled Number or the Destination Number. The data conflict filter at the moment differentiates two kinds of usage of this data, *writing/modification* and *usage for decision*.

The data conflict filter recognises two kinds of conflicts on this data, a *write-after-use* conflict, where data is modified that has been already used for a decision that possibly has influenced the progress of the call, and a *use-after-write* conflict, where modified data is used for decision. It has to be noted that in cases where the triggering order cannot be determined satisfactorily, the filter acts defensively, i. e., as if both orders may occur.

2.3.3 Resource conflicts

The resource filter detects conflicting access to common resources in the call scenario, like, e. g., the voice channel, the #-key or the hook-flash signal. The model for resource usage currently applied by this filter is very simple and will have to be improved if it shows to be practically usable.

A resource can be *allocated* by a service. This means that this service assumes exclusive access to the resource. This is, e. g., the case for the Call Waiting (CW) service, where this service assumes exclusive access to the voice channel to apply the CW signal at any time, or in the CCC example of Cameron [9], where the CCC service claims exclusive usage of the #-key. A resource can also simply be *used* by a service. If a resource is used or allocated that already has been allocated by another service, a conflict is detected by the filter and reported.

2.4 The triggering order problem

For the application of some of the criteria described above, knowledge of the triggering order of the services is needed. In formal verification approaches, this information is obtained implicitly, e. g., by executing a state space exploration for the services that are described in a formal specification language. In the approach described here, this precedence is detected in a different way, as there is no complete formal description of the services and no state space exploration component included.

If the initial triggers of the two services use the same Detection Point of the same BCSM, it is currently assumed that there is no explicit precedence rule. As a consequence, this case is treated as if it was possible to trigger the services in both orders.

If the triggers of the two services use different Detection Points of the same BCSM or of a pair of two associated O- and T-BCSMs, a *precedence matrix* is used in order to find out which one of the services is triggered first. This precedence matrix is a $N \times N$ matrix of boolean values, if N is the total number of detection points in the O- and T-BCSM. Detection Point j can be reached after DP i, if $m_{ij} = \text{true}$. This matrix has been calculated using a LOTOS description of the BCSMs and the CÆSAR/ALDÉBARAN toolkit and is provided as a separate configuration file to the trigger precedence filter.

Cases in which the triggers are situated within different associations of O- and T-BCSMs are more difficult to treat. In such cases, the causal relationships between IN services and the establishment of subsequent call legs (associations of O- and T-BCSMs) can be exploited. There is always at least one trigger detection point in the first leg of a call scenario. As the creation of subsequent legs is initiated by this service of the first leg, the triggering order is clear due to this causal relationship.

At the moment, there seems to be no simple solution to take into account phenomena that are caused if a service does not process the BCSMs in the order given in the IN standards, but “jumps” arbitrarily within these models. In order to cope with that kind of services, it seems to be inevitable to include a reachability computation component into the tool, however, the computational overhead and thus the processing time would increase significantly because of the additional complexity of such an analysis compared with the static reachability matrix used in the current version.

3 A tool prototype for filtering interaction-prone scenarios

3.1 General concepts

Based on the criteria described in the previous section, a tool prototype has been implemented. Its design is based on the principles of modularity, extensibility, and configurability, as it is planned to use this tool as a platform for the quick and simple evaluation of new interaction detection criteria and algorithms. The principal structure of the tool is shown in Fig. 2. The tool consists of four major parts:

The *initialisation component* is responsible for selecting and parsing the service descriptions, which are provided in a way described in section 3.2. From pairs of these service descriptions – which represent the pairs of services to be examined for potential interactions – a list of call scenarios is created and restricted as described in section 2.2.2. This list is central to the further filtering steps and is therefore stored in the *call scenario repository* in order to make it available to the subsequent processing steps.

The *filtering component* of the tool is highly modular and configurable by a filter configuration file and additional auxiliary files like the precedence matrix described in section 2.4. All filters are designed using the same interface, which eases addition or removal of a filter. Two types of filters are defined, mandatory filters which are always active and optional filters that can be activated or deactivated by commands in the filter configuration file.

The *result generation component* creates a file reporting the results of the analysis. In the current version of the tool prototype, this is a simple text file. In later versions, this probably will be replaced by a more sophisticated post-processing that allows better navigation and overview.

As the criteria applied by the tool are informal, and also the methods available to find out whether an interaction is possible or not are much less strict than in the application of formal methods, there is the danger of false negative[3] results. Respect has been paid to this problem by following a *pessimistic* approach, i. e. in cases where it is not clear what happens, for example if the trigger precedence cannot be detected clearly, it is assumed that both precedence may be possible. Thus, the results tend to be worse than reality.

3.2 Service scenario description

For service scenario description, a very simple description language has been designed, which is interpreted by the tool. This language only comprises the static properties

[3] “negative” in this context denotes the absence of possible interactions

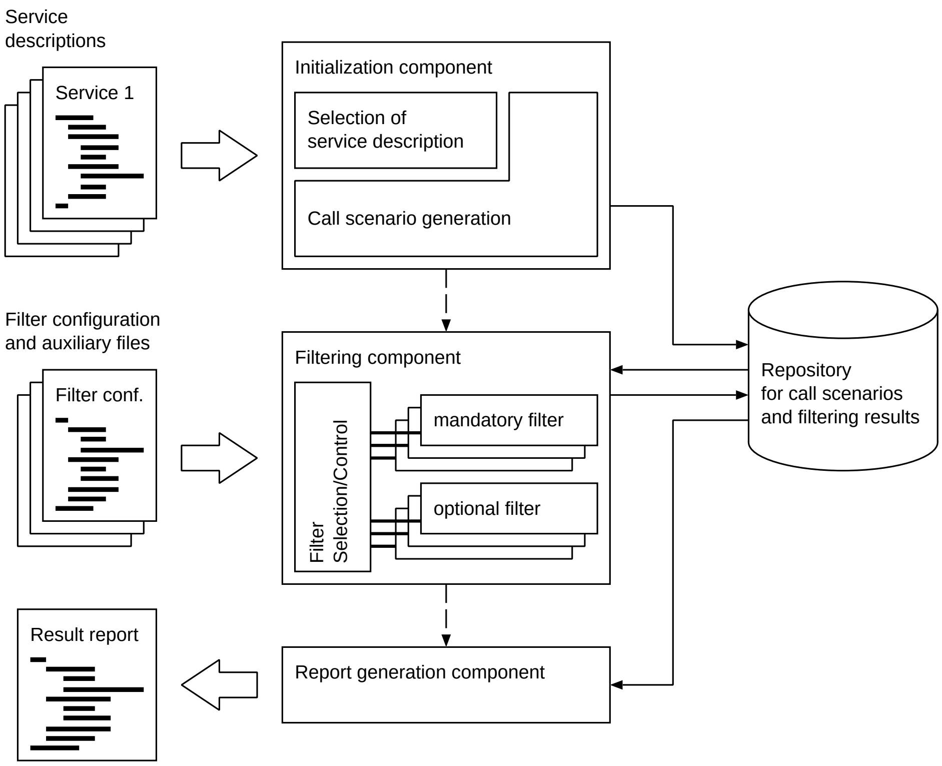


Figure 2: Tool structure.

of a service scenario, and no information on its dynamic behaviour. For each service, it contains the name of the service, which is also used in the result report, and the number of formal participants that belong to the service scenario. Each of these formal participants has got a number and a name, that is also used in the result report. For each of the formal participants, the BCSMs involved are listed, with their detection points used, the type and additional criteria of the detection points, and the resources and data used or modified during processing of this detection point. In the final section of the description, the topology of the scenario is described, i. e., how the formal participants (and their BCSMs) are connected to each other.

In Table 1, all relevant grammatical productions for the service scenario description language are listed. It has to be emphasised that the *Resource* and *Data* productions in this table are incomplete. This is a consequence of the extensibility of the tool, that allows to specify additional resources and data and to define appropriate actions on these. Table 2 shows the application of this description language for a description of a Terminating Call Screening (TCS) service.

4 Case studies

The case studies presented in this section aim at giving an idea of how the tool operates. Therefore, the example services chosen have not been selected because of their practical relevance or their novelty, but because of their suitability to demonstrate the operation of the tool prototype.

FeatureDescription:=	FEATURE (*NoOfFormalParticipants* , *FeatureName*); BEGIN *FormalParticipantList* END BEGIN *FeatureLegList* END
FormalParticipantList :=	*FormalParticipant* | *FormalParticipant FormalParticipantList*
FormalParticipant :=	FORMAL_PARTICIPANT (*FormalParticipantNumber* , *FormalParticipantName*); *BcsmList*
BcsmList :=	*Bcsm* | *Bcsm BcsmList*
Bcsm :=	BCSM (ORIGINATING); *DpList* | BCSM (TERMINATING); *DpList*
DpList :=	*Dp* | *Dp DpList*
Dp :=	DP (*DpNumber*); *DpPropertyList*
DpPropertyList :=	*Trigger ResourceList DataList*
Trigger :=	TRIGGER (*TriggerType* , *TriggerCriterion*);
TriggerType :=	TDP_R | TDP_N | EDP_R | EDP_N | EMPTY
TriggerCriterion :=	LINE | GROUP | OFFICE | EMPTY
ResourceList :=	*Resource* | *Resource ResourceList*
Resource :=	RESOURCE (*ResourceName* , *ResourceOperation*);
DataList :=	*Data* | *Data DataList*
Data :=	DATA (*DataName* , *DataOperation*);
FeatureLegList :=	*FeatureLeg* | *FeatureLeg FeatureLegList*
FeatureLeg :=	LEG (*FormalParticipantIdentifierOfOBcsm* , *FormalParticipantIdentifierOfTBcsm*);

Table 1: Grammatical productions of the feature description language

```
FEATURE (2, 'Terminating Call Screening (TCS)');      /* Service scenario header       */
BEGIN                                                 /* PARTICIPANT SECTION           */
  FORMAL_PARTICIPANT(1, 'Caller');                    /* "A" party                     */
    BCSM(ORIGINATING);                                /* O-BCSM of "A" party           */
      DP(EMPTY);                                      /* no DP armed (no trigger)      */
      TRIGGER(EMPTY,EMPTY);                           /* trigger type not used         */
      RESOURCE(EMPTY, EMPTY);                         /* no resource used              */
      DATA(EMPTY, EMPTY);                             /* no data used                  */
  FORMAL_PARTICIPANT(2, 'Called Party (Screening)');  /* "B" party (with TCS)          */
    BCSM(TERMINATING);                                /* T-BCSM of "B" party           */
      DP(12);                                         /* DP12 is armed as trigger      */
      TRIGGER(TDP_R, EMPTY);                          /* DP type is TDP-Request        */
      RESOURCE(EMPTY, EMPTY);                         /* No resources occupied         */
      DATA(CallingLineIdentity, ASSERT);              /* CLI used for decision         */
END                                                   /* PARTICIPANT SECTION END       */
BEGIN                                                 /* TOPOLOGY SECTION              */
  LEG(1,2);                                           /* O-BCSM of "A" / T-BCSM of "B" */
END                                                   /* TOPOLOGY SECTION END          */
```

Table 2: Exemplary service scenario description of Terminating Call Screening (TCS)

4.1 Results of topological filtering

If IN services are regarded, there is a limited number of possible service scenario topologies that usually belong to the following three simple types:

- Originating Call Screening type (OCS type) with a pair of associated O- and T-BCSMs, the trigger in the O-BCSM,
- Terminating Call Screening type (TCS type) like OCS type, except the trigger is in the T-BCSM (see Fig. 1a),

- Call Forwarding type (CFW type), with two pairs of associated O- and T-BCSMs, the creation of the second pair initiated by the service triggered by a detection point in the T-BCSM of the first pair (see Fig. 1b).

The service scenarios of services like Conference Call and Call Waiting are currently not covered by the tool to a full extent, although the topological criteria hold as well for this kind of services. Table 3 shows the number of call scenarios obtained by application of the topological conditions for the basic service scenario types. The first number in the results section denotes the total number of topologically different call scenarios in a specific service combination, the second one the number of non-problematic and the third one the number of problematic scenarios (see section 2.2.4).

	different services			same service
	OCS type	TCS type	CFW type	(symmetry is exploited)
OCS type	15 / 1 / 1	15 / 1 / 1	52 / 2 / 7	11 / 1 / 1
TCS type	*not relevant*	15 / 1 / 1	52 / 2 / 7	11 / 1 / 1
CFW type	*not relevant*	*not relevant*	203 / 6 / 34	117 / 4 / 20

n_{top} / $n_{nonprob}$ / n_{prob} with n_{top} denoting the number of topologically different call scenarios, $n_{nonprob}$ denoting the number of non-problematic call scenarios, where each of the two services' participants are mapped to different actual participants, and n_{prob} denoting the number of problematic scenarios, where at least two different formal participants of one service are mapped to the same actual participant.

Table 3: Number of call scenarios obtained for services of different basic topology types by application of the topological conditions.

In order to demonstrate the results obtained by application of the tool prototype to a larger set of services, a very small test suite of six different services has been selected and analysed by the tool with respect to topological conditions, trigger reachability, and manipulation of the *DestinationNumber* Information Element. The results are displayed in Table 4 and have to be interpreted as follows: In the CFU/CFU example, 2 of the non-problematic and 12 of the problematic scenarios exhibit trigger collisions, i. e. the services are triggered at the same DP. In section 4.3, exactly these cases will turn out to be the irrelevant ones. In the CFB/CFNR example, 2 of the non-problematic cases have got mutually unreachable triggers.[4] In the CFB/OCS example, one of the non-problematic scenarios has a *write-after-use* situation, one a *use-after-write* situation. All 7 of the problematic scenarios also have one of these situations, however, it cannot be decided which one because of problems determining the triggering order in these topologies.

In the following two sections, two specific service combinations are examined in more detail in order to provide a deeper insight into the operation of the tool prototype.

[4]This unreachability can be detected by the tool using the reachability matrix described in section 2.4. The absence of unreachability does not imply that the triggers are mutually reachable, as reachability detected by this tool only is a necessary condition for reachability in reality.

<table>
<tr><th></th><th colspan="2">Criteria</th><th>CFB</th><th>CFNR</th><th>CFU</th><th>FREER</th><th>OCS</th><th>TCS</th></tr>
<tr><td rowspan="5">CFB</td><td rowspan="2">Trigger</td><td>Collisions</td><td>2 / 12</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Unreachability</td><td></td><td>2 / 0</td><td></td><td></td><td></td><td></td></tr>
<tr><td rowspan="3">DestNo</td><td>write-after-use</td><td></td><td></td><td></td><td></td><td>1 / 0</td><td></td></tr>
<tr><td>use-after-write</td><td></td><td></td><td></td><td></td><td>1 / 0</td><td></td></tr>
<tr><td>not decidable</td><td></td><td></td><td></td><td></td><td>0 / 7</td><td></td></tr>
<tr><td rowspan="5">CFNR</td><td rowspan="2">Trigger</td><td>Collisions</td><td rowspan="5">not relevant</td><td>2 / 12</td><td></td><td></td><td></td><td></td></tr>
<tr><td>Unreachability</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td rowspan="3">DestNo</td><td>write-after-use</td><td></td><td></td><td></td><td>1 / 0</td><td></td></tr>
<tr><td>use-after-write</td><td></td><td></td><td></td><td>1 / 0</td><td></td></tr>
<tr><td>not decidable</td><td></td><td></td><td></td><td>0 / 7</td><td></td></tr>
<tr><td rowspan="5">CFU</td><td rowspan="2">Trigger</td><td>Collisions</td><td colspan="2" rowspan="5">not relevant</td><td>2 / 12</td><td></td><td></td><td>1 / 5</td></tr>
<tr><td>Unreachability</td><td></td><td></td><td></td><td></td></tr>
<tr><td rowspan="3">DestNo</td><td>write-after-use</td><td></td><td></td><td>1 / 0</td><td></td></tr>
<tr><td>use-after-write</td><td></td><td></td><td>1 / 0</td><td></td></tr>
<tr><td>not decidable</td><td></td><td></td><td>0 / 7</td><td></td></tr>
<tr><td rowspan="5">FREER</td><td rowspan="2">Trigger</td><td>Collisions</td><td colspan="3" rowspan="5">not relevant</td><td>1 / 1</td><td>1 / 1</td><td></td></tr>
<tr><td>Unreachability</td><td></td><td></td><td></td></tr>
<tr><td rowspan="3">DestNo</td><td>write-after-use</td><td></td><td></td><td></td></tr>
<tr><td>use-after-write</td><td></td><td></td><td></td></tr>
<tr><td>not decidable</td><td></td><td>1 / 1</td><td></td></tr>
<tr><td rowspan="5">OCS</td><td rowspan="2">Trigger</td><td>Collisions</td><td colspan="4" rowspan="5">not relevant</td><td>1 / 1</td><td></td></tr>
<tr><td>Unreachability</td><td></td><td></td></tr>
<tr><td rowspan="3">DestNo</td><td>write-after-use</td><td></td><td></td></tr>
<tr><td>use-after-write</td><td></td><td></td></tr>
<tr><td>not decidable</td><td></td><td></td></tr>
<tr><td rowspan="5">TCS</td><td rowspan="2">Trigger</td><td>Collisions</td><td colspan="5" rowspan="5">not relevant</td><td>1 / 1</td></tr>
<tr><td>Unreachability</td><td></td></tr>
<tr><td rowspan="3">DestNo</td><td>write-after-use</td><td></td></tr>
<tr><td>use-after-write</td><td></td></tr>
<tr><td>not decidable</td><td></td></tr>
<tr><td colspan="9">All numbers given are partitioned into non-problematic / problematic call scenarios.</td></tr>
</table>

Table 4: Results of the application of the tool prototype to a small test suite consisting of six services – CFB (Call Forwarding on Busy), CFNR (Call Forwarding on No Reply), CFU (Call Forwarding Unconditional), FREER (Freephone Routing), OCS (Originating Call Screening), and TCS (Terminating Call Screening). The calculation of these 21 service combinations required 5.5 CPU seconds on a HP PA-RISC workstation.

4.2 Call Forwarding Unconditional with Announcement (CFU-A) and Terminating Call Screening with Announcement (TCS-A)

CFU-A is based on the well known call forwarding service, the service scenario of which is shown in Fig. 1b. In this example, the service is enhanced with an announcement ('This call is being forwarded to C') played to A before the forwarding is initiated. TCS-A is a call screening service triggered at the T-BCSM, that also plays an announcement in case the call is rejected.

According to equation (1), for the total of 5 formal participants involved in these two services, a number of $k_5 = 52$ topologically different call scenarios can be established. Using the condition of section 2.2.3, this number can be reduced to 9 call scenarios. Application of the criterion described in section 2.2.4 leads to 2 call scenarios that are not problematic (i. e., do not contain formal participants of one service that are mapped to the same actual participant), and to 7 problematic scenarios. These results are illustrated in Fig. 3.

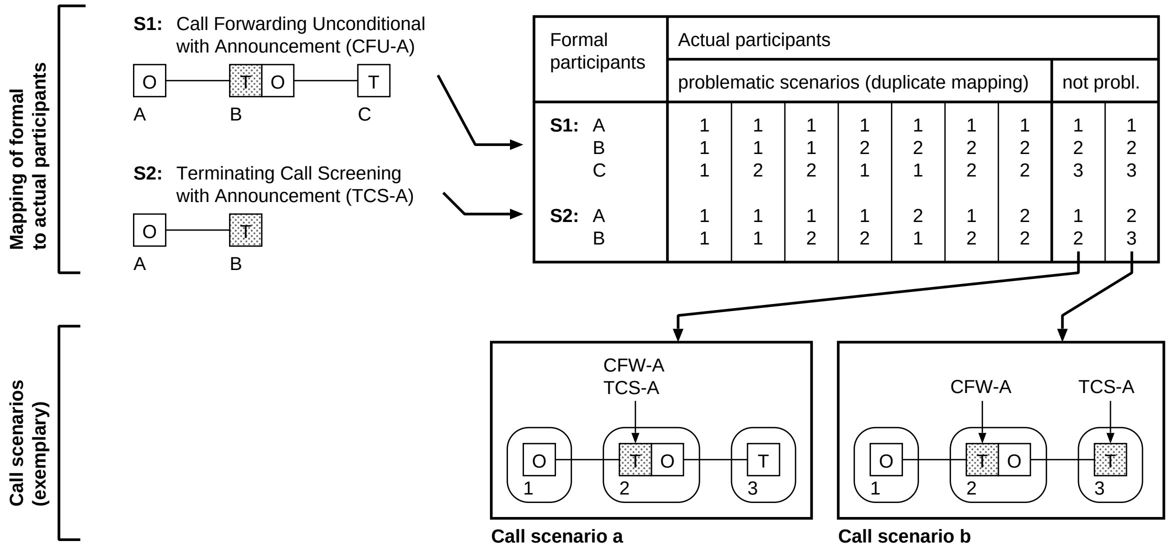


Figure 3: Case study of CFU-A and TCS-A.

An application of the filters described in section 2.3 leads to the following results: As both services are initially triggered on the same detection point (DP 12, "Termination Attempt Authorised"), a *trigger collision* is detected for call scenario *a* in the lower part of figure 3. The triggering order in this case is considered as undefined by the tool. For call scenario *b*, no trigger collision occurs due to the allocation of the triggers in different BCSMs. The triggering order is determined by the causal relationship between the service triggered in the first leg (CFU-A) and the establishment of the forwarding leg.

The next filter that is applied checks for *data conflicts*. In the service definitions of this cases study, CFU-A changes the value of the Calling Number. TCS-A, however, checks this value and decides how to proceed with this call based on it. As the triggering order in call scenario *(a)* cannot be determined, both orders are examined and consequently a *write-after-use* as well as a *use-after-write* conflict on the Calling Number are detected. In call scenario *(b)*, it is clear that CFU-A is triggered before TCS-A, thus a use-after-write conflict on the Calling Number is detected. The CFU-A also modifies the Called Number, however, as this data is not used by TCS-A, no conflicts occur between the two services.

In the simple model currently used for detection of *resource collisions*, no collision is detected, as both services only *use* the voice channel resource for a single announcement. This usage does not occur at the same time, as the services are not active (triggered) simultaneously.

4.3 Two instances of Call Forwarding Unconditional (CFU)

This example has been selected as it is topologically interesting. The condition of section 2.2.3 leads to 24 call scenarios, of which 4 turn out to be non problematic. A manual inspection of these call scenarios provides the insight that some contain the same CFU service being triggered at the same DP of the same BCSM, which clearly is not sensible. The remaining call scenarios are 2 with non-problematic topologies (Fig. 4a) and 8 with problematic topologies (Fig. 4b). The problematic topologies cover all the cases of forwarding to oneself and mutual forwarding.

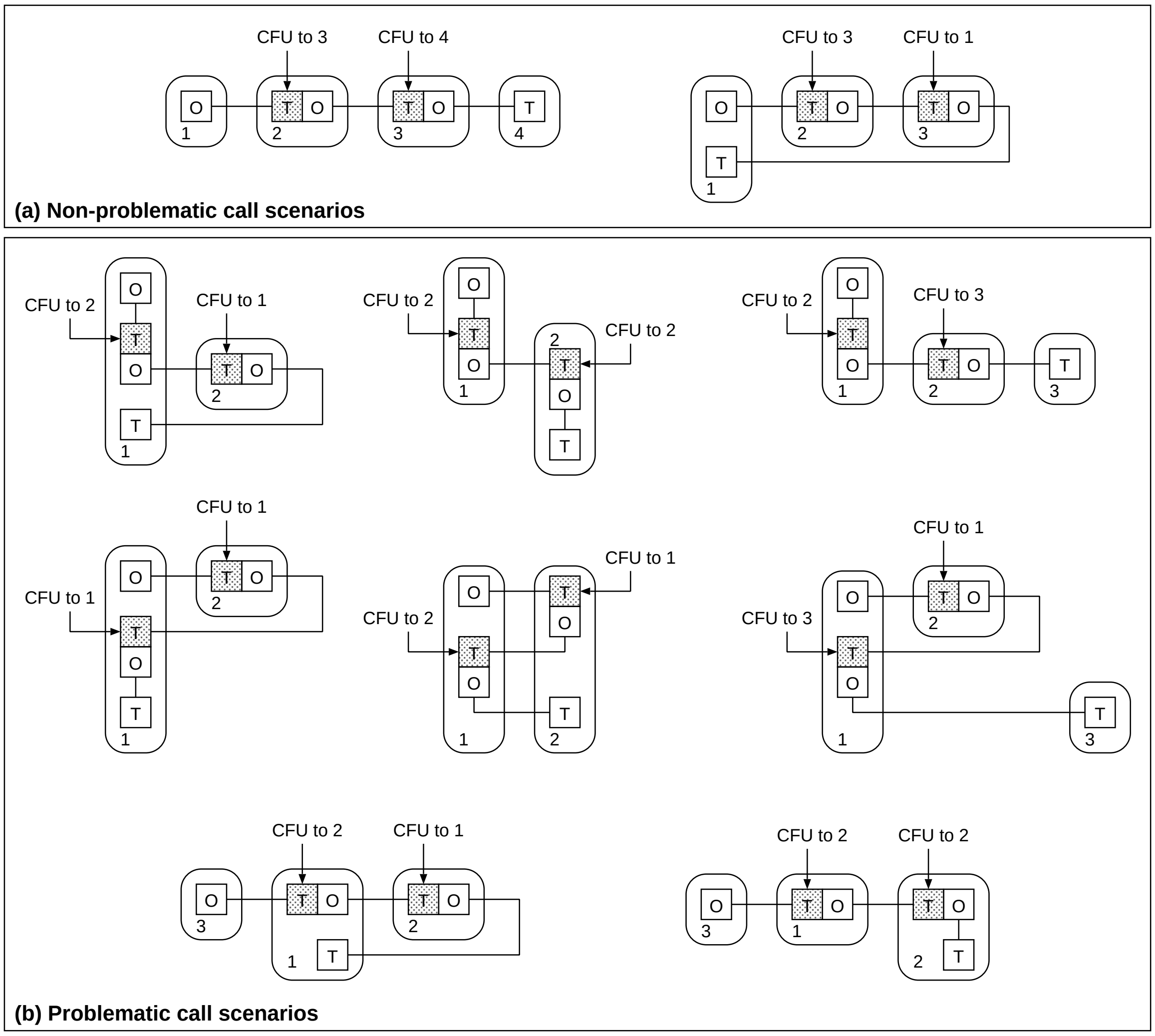


Figure 4: Non-problematic and problematic call scenarios of the case study of two instances of CFU.

5 Related Work

In their formal verification approach, Au and Atlee [10] also have used data, resource, and assertion based criteria for interaction detection. However, they apply these criteria in a state space exploration approach, where they report of explosion problems, while in this contribution, similar criteria are applied in an informal (and less precise) approach. A combination of such approaches, in the sense of a precise investigation after a coarse filtering could be valuable.

There is only little tool support reported in the literature that could be counted to a class of approaches in the sense of Kimbler's filtering approach [1]. In their building block approach, Lin and Lin [11] have presented a front-end to a formal verification tool, that has some similarities to the tool presented here. In contrast to that tool, the tool presented here is currently not a front-end for a formal verification tool, but uses informal criteria that are easy to establish and extremely fast to apply. An additional feature of the tool is the establishment of call scenarios with real participants, which is not done in that approach.

There is also some methodological parallelism to the work of Kuisch *et al.* [12], who apply a "template that exactly highlights those aspects of the service that are relevant for its interaction with other services." However, in their work, only service that are triggered in the same BCSM instance are regarded. Also, there seems to be no automatic tool support for their approach. Nevertheless, a number of their ideas have been used in the tool described in this contribution.

Cameron *et al.* [13] describe a tool called FIMAMP, that is able to classify and group features with similar properties into feature groups. With this approach, they manage to reduce the combinatorial complexity of the interaction problem significantly, as the now have to analyse compositions of feature types instead of features.

6 Conclusions

The tool prototype presented in this contribution is able to determine a set of interaction-prone scenarios from simple IN service descriptions containing only static structural information on the services, hiding most of the implementation or dynamic aspects. The identification of these critical scenarios is based on a number of relatively simple criteria, like conflicts in the use or modification of data, or resource conflicts. This is demonstrated by some simple case studies.

The value of this tool lies in its ability to pick the most interesting, i. e., most interaction-prone scenarios from the huge number of possible scenarios that exist if IN services are used in combination. This selection is cheap in terms of processing as well as in terms of IN service description effort. However, it does not replace a thorough analysis using other methods (like FDTs, formal verification, testing) of the critical scenarios, as only a pre-selection based on a simplified service specification is carried out.

As this contribution reports on ongoing research, there are still some open issues. Especially the handling of topologically more complex services like conference calls has to be studied. Also, many extensions in the modelling and evaluation of resource and data usage have to be made.

References

[1] K. Kimbler. Addressing the interaction problem at the enterprise level. In P. Dini, R. Boutaba, and L. Logrippo, editors, *Feature Interactions in Telecommunication Networks IV*, pages 13–22. IOS Press, Amsterdam, June 1997.

[2] J. Bredereke. Detection of feature interactions in intelligent networks by verification. *Software-Concepts and Tools*, 17(3):121–139, 1996.

[3] Distributed functional plane for Intelligent Network CS-1. ITU-T Recommendation Q.1214, Geneva, 1993.

[4] T. Magedanz and R. Popescu-Zeletin. *Intelligent Networks: Basic Technology, Standards and Evolution.* International Thomson Computer Press, London, 1996.

[5] I. Faynberg, L. R. Gabuzda, M. P. Kaplan, and N. J. Shah. *The Intelligent Network Standards: their application to services.* McGraw-Hill, New York, 1997.

[6] D. O. Keck. Identification of call scenarios with potential feature interactions. In T. Margaria, editor, *Proceedings of the International Workshop on Advanced Intelligent Networks (AIN'96)*, MIP-9604, pages 42–55. Universität Passau, Fakultät für Mathematik und Informatik, March 1996.

[7] J. Riordan. *An Introduction to Combinatorial Analysis.* John Wiley & Sons, New York, 1958.

[8] M. Jeger. *Einführung in die Kombinatorik – Band 1.* Ernst Klett Verlag, Stuttgart, 1973.

[9] E. J. Cameron, N. D. Griffeth, Y. J. Lin, M. E. Nilson, W. K. Schnure, and H. Velthuijsen. A feature interaction benchmark for in and beyond. In L. G. Bouma and H. Velthuijsen, editors, *Feature Interactions in Telecommunications Systems*, pages 1–23. IOS Press, Amsterdam, May 1994.

[10] P. K. Au and J. M. Atlee. Evaluation of a state-based model of feature interactions. In P. Dini, R. Boutaba, and L. Logrippo, editors, *Feature Interactions in Telecommunication Networks IV*, pages 153–167. IOS Press, Amsterdam, June 1997.

[11] F. J. Lin and Y. J. Lin. A building block approach to detecting and resolving feature interactions. In L. G. Bouma and H. Velthuijsen, editors, *Feature Interactions in Telecommunications Systems*, pages 86–191. IOS Press, Amsterdam, May 1994.

[12] E. Kuisch, R. Janmaat, H. Mulder, and I. Keesmaat. A practical approach to service interactions. *IEEE Communications Magazine*, 31(8):24–31, August 1993.

[13] J. Cameron, K. Cheng, F. J. Lin, H. Liu, and B. Pinheiro. A formal AIN service creation, feature interactions analysis and management environment: An industrial application. In P. Dini, R. Boutaba, and L. Logrippo, editors, *Feature Interactions in Telecommunication Networks IV*, pages 342–346. IOS Press, Amsterdam, June 1997.