Detecting Feature Interactions in the Terrestrial Trunked Radio (TETRA) Network using Promela and Xspin [⋆]

Carl B. Adekunle and Steve Schneider

Department of Computer Science,
Royal Holloway, University of London,
Egham Hill, Egham,
Surrey, TW20 0EX,
United Kingdom.
Tel: +44 (0)1784 443912
Fax: +44 (0)1784 439786
email: `c.adekunle@dcs.rhbnc.ac.uk`

Abstract. The problem caused by feature interactions serve to delay and increase the costs of introducing features to existing networks. We introduce a three stage technique that detects interactions in the TETRA network. Promela is used to model the network and features and its requirements are specified using linear temporal logic. The Xspin toolkit is used to verify and validate the model against its requirements.

1 Introduction

Telecommunications networks comprise numerous features that add extra functionalities on top of the basic services provided. With new features being incrementally added, it is unclear how they interact. For example, will the addition of a feature interfere with the behaviour of existing features? Or perhaps the new feature demonstrates unexpected behaviour as a result of integration with existing features. The problem caused by these feature interactions serve to delay and increase the costs of introducing features to existing networks. The problem is exacerbated because many features are designed and implemented by different companies across global networks. Assumptions are made about network implementations and consequently the behaviour of features differ. The feature interactions field [KK98] attempts to avoid, detect and resolve interactions at various stages of the features life cycle.

We research feature interactions in Terrestrial Trunked Radio [1] (TETRA)[TR95], which is a digital private mobile radio network specified by the European Telecommunications Standard Institute (ETSI). In TETRA, features are referred to as supplementary services though, for the purpose of this paper, we use the term features.

The TETRA network and features are modelled using the Protocol Meta language (Promela) [Hol90] and we specify requirements that the model should satisfy using

⋆ Research funded by Motorola.

[1] Formally Trans-European Trunked Radio.

linear temporal logic [MP91]. The language of temporal logic is used to specify and reason about properties such as safety and liveness at an abstract level. We use the Xspin toolkit to verify the model for general properties such as deadlock and to validate the model against its requirements.

In this paper, we present and exploit a three stage technique to detect feature interactions. The technique is general enough to be applied to other languages and toolkits. It relies on the validation of requirements being completed for the first two stages to detect interactions in the final stage.

The rest of the paper is organised as follows: section 2 discusses feature interactions in more detail and outlines the three approaches used to tackle the problem. Section 3 introduces TETRA and highlights the areas of the network analysed for interactions. It also discusses the information extraction process used to decide what operations to model. In section 4 we summarise why we use Promela and Xspin to model the TETRA network and features. We also give an overview of temporal logic and the types of requirements it can be used to specify. The technique used to detect feature interactions is presented in section 5 and we demonstrate its effectiveness in section 6. Section 7 presents our conclusion.

2 Feature Interactions

Features in telecommunication networks add extra functionality on top of the basic services provided. Examples of features are Call Forwarding Unconditional (CFU) [Eura] and Barring of Incoming Calls (BIC) [Eurb]. Call Forwarding Unconditional allows its subscriber to divert all incoming setup requests to a designated mobile other than itself. Barring of Incoming Calls screens all incoming setup requests to the subscriber against a barred list. If the calling mobile is on that list, it is prevented from connecting to the subscriber. When both features are invoked by their subscriber, it is not clear what should happen to an incoming setup request from a calling mobile that is on the barred list. For example, mobile C subscribes to both features and has mobile B as its divert for the CFU and mobile A as barred on the BIC. Mobile A then makes a setup request to join C, should the CFU forward mobile A to B or should A be barred from not only connecting to C but also to B? Such uncertainty is a contributing factor in feature interactions.

The feature interactions field has arisen as a result of the need to detect and resolve interactions using three approaches, which are:

Avoidance - Involves using an architecture to prevent interaction by tackling issues such as resource contention. The features are specified, implemented and tested within the architecture [JP98].

Detection - Employs the use of model checking techniques such as simulation, verification and validation to detect interactions [LL94].

Resolution - Focuses on run time detection of interactions [TM98]. Feature interactions managers (FIM) are often used to resolve the interactions by prioritising the activation of features.

No single approach can detect all interactions, so a combination of all three provides the best possibility of detecting and resolving interactions during the software cycle of a feature.

A more generalised definition of interactions is: The behaviour of a feature is suppressed or modified by the existence of another feature. In our research, feature interactions is defined as the behaviour of a feature preventing another from satisfying its requirements. We use a detection approach to detect interactions using verification and validation model checking.

3 Terrestial Trunked Radio (TETRA)

TETRA is the next generation of digital private mobile radio (PMR) network. It allows the exchange of voice and data between mobile radios in a group and is currently scheduled to replace existing analog networks used by organisations like the police.

To connect to the network, a mobile radio requests call setup to another in its group via the Switching and Management Infrastructure (SwMI), which is made up of base stations, switches and controllers. The called mobile accepts or rejects the request by informing the SwMI which in turn informs the calling mobile of the response. Acceptance of a call setup request ensures both the calling and called mobiles are connected to the group. To transmit voice or data in the group, a mobile has to request permission from the SwMI, which grants or rejects the request. If granted, all mobiles in the group are informed of the identity of the mobile and all receive the transmission.

3.1 Circuit Mode Control Entity

In this section, we provide an overview of the areas related to mobile communication. The European Telecommunication Standard (ETS)[TR95] specifies the services and protocol used by the Circuit Mode Control Entity (CMCE), which lies in layer three of the International Organisation for Standardisation (ISO), Open Systems Interconnection (OSI) seven layer reference model.

Each hand held or vehicle based mobile has one CMCE that is responsible for connecting it to a group call. Figure 1 shows the four sub-entities that make up the CMCE. The dashed lines represent sub-entities and bi-directional channels that are not necessary to our research, so we focus only on the Call Control (CC) and Protocol Control (PC) sub-entities as they are essential to providing basic bearer services like call connections and transmission of voice and data.

The Call Control sub-entity is responsible for initiating, maintaining and releasing calls made by the mobile. It communicates via channel *ra* to user applications [2] (UA) that can be anything from a press to talk button to a liquid crystal display on the mobile. User applications are responsible for informing Call Control of activities such as call setup request to another mobile. Channel *rd* connects Call Control with the Protocol Control sub-entity, which is responsible for error correction and communicating with the SwMI using channel *ri*[3].

[2] User Applications are not defined by the ETS and therefore not shown in the figure.

[3] Via the Mobile Link Entity, which is not shown in the figure and is not modelled.

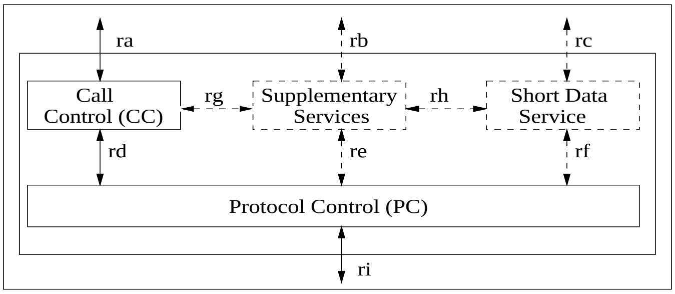

Fig. 1. Circuit Mode Control Entity

3.2 Information Extraction Process

In this section, we highlight the process we use to extract information from the ETS to model the TETRA network. The aim is to derive the minimum amount of information to successfully model the basic bearer services.

As mentioned in the previous section, only the Call Control and Protocol Control sub-entities from the CMCE need to be included in the model as processes. Though the Supplementary Services sub-entity is responsible for configuring and invoking features on behalf of the mobile, there is little need to model it as all its responsibility can be explicitly modelled in the SwMI process. The Short Data Services sub-entity has no bearing on the features, therefore it is not modelled. As the SwMI and user applications are outside the scope of the ETS, we model them as processes using cross referenced information from other sources.

All communication in TETRA is done using protocol data units (PDUs). The structure of each PDU varies according to its primitive type, which identifies the PDU. For example, the primitive *tncc_setup_request* has up to twenty four parameters. We simplify the structure of PDUs by requiring all primitives to have only four parameters as shown in figure 2. The front of the PDU is occupied by its primitive type. The four parameters identify the mobile (*Tsi*), its group (*GTsi*), the contact mobile (*DestTsi*) and any additional information about the PDU (*Msg*). To cut down on the number of PDUs in the model, we do not include call progress or query information. We also identify the variables that need to be included in the model.

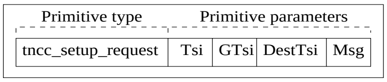

Fig. 2. Protocol Data Unit Structure

We decided to model three mobiles, as this number is sufficient for our analysis. Figure 3 provides a diagrammatic view of the ten processes and bidirectional channels that make up the TETRA model. The behaviour of each mobile is represented by the *ua*, *cc* and *pc* processes.

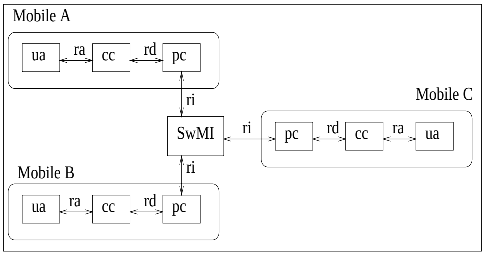

Fig. 3. Processes in the TETRA Model

4 Why Promela And Xspin?

The Specification and Description Language (SDL) was introduced in the late 1970's by the International Telecommunication Union (ITU)[4] and has found popular use in the telecommunications industry. Its graphical syntax and comprehensive toolkit support has contributed to it being the language of choice when implementing TETRA network. However, for our research it is unsuitable mainly because its semantics [SDL] allows processes to discard messages at the head of their FIFO queues [5], when they cannot remove messages due to absent corresponding input symbols[6]. The effect of this implicit consumption is to make it virtually impossible for an SDL system to deadlock due to processes being unable to remove messages from their queues.

As Promela uses a C like text syntax, we found engineers familiar with programming languages required little adjustment in understanding and using it. Promela's use of never claims to represent temporal formulae means that requirements could be specified and validated using the Xspin toolkit. Though we use Promela and Xspin, all results need to be applicable to SDL. How this is done is outside the scope of this paper.

4.1 Linear Temporal Logic

Using linear temporal logic, it is possible to specify and reason about temporal properties at an abstract level. A sequence of infinite states is regarded as a computation. The sequence represents a single path starting with an initial state s_0 and on execution of a transition, progresses to a successor state shown as $\sigma : s_0, s_1, s_2, \cdots$. Each state in the

[4] Formally CCITT.

[5] The SDL term is signal paths.

[6] SDL does have a SAVE symbol to store messages that can be handled in later states. If this operator is not used, the message is discarded.

sequence has an associated successor. First order temporal formulae are derived from combining state formulae with boolean connectives and future and past temporal operators. As the past operators do not add any expressibility to the language of temporal logic, we use only the future operators $\Box$ (*always*), $\Diamond$ (*eventually*), $\mathcal{U}$ (*until*) and $\mathcal{W}$ (*unless*). We do not cover the $\bigcirc$ (*next*) operator as it is the only operator able to detect stuttering (i.e. when a state is repeated in succession in a sequence). The definition of future operators is based on the temporal formula holding at states in the computation.

Specifying Requirement Properties. Using the temporal operators, we can specify properties of interest such as safety. For example, specifying the value of variable $count$ is never less than zero $\Box(count \geq 0)$. Liveness requires the system to eventually carry out a desirable behaviour such as system termination for a scheduled downtime $\Diamond(terminate)$. A persistence property specifies that there are finitely many positions where a property does not hold, but when the property occurs it holds indefinitely. For example, on startup a system will reach a stable state and maintain the stability $\Diamond\Box(system\_stable)$. A response property allows the specification that an action has a corresponding reaction, either at the same position or in the future. For example, every request has a corresponding response $\Box(request \rightarrow \Diamond response)$. Other properties that can be specified as temporal formulae are obligation and reactivity.

5 A Technique to Detect Feature Interactions

Building and verifying the TETRA model is not sufficient to detect feature interactions. There is a need to apply the modelling language and toolkit in a manner that increases the likelihood of detecting interactions. We devised and use a three stage technique that is general enough to be used with any language and toolkit that allows validation of requirements. Each stage consists of iterative steps. The first two stages apply to most system development and do not involve the detection of feature interactions. However the two are essential to the final detection stage, which uses requirements to detect feature interactions. Verification and Validation is preferred over simulation because they explore more computations and check for general properties such as deadlock and livelock. The rest of this section, presents an overview of the stages involved in the technique.

5.1 Stage One

This stage is concerned with constructing the requirements for the base model, implementing, verifying and validating the base model against its requirements. The following steps are applied:

1. Constructing the base model requirements (REQ):
   REQ normally consists of a number of requirements, which are constructed using information from the information extraction process. This ensures the requirements are based on the intended behaviour of the model. For instance, there is no point in having a requirement about performance when the model is implemented without any performance operations.

2. Model the system ($Stage1\ model$): Language dependent.
3. Verify $Stage1\ model$ for general properties:
   On discovery of an error, resolve it and repeat from this step. When all errors are resolved proceed to the next step.
4. Validate $Stage1\ model$ against REQ:
   If $Stage1\ model$ does not satisfy REQ and provided REQ is correctly constructed, resolve the error and repeat from step three. If REQ is incorrectly constructed, use the validation result to modify REQ and repeat this step. The aim of this step is to have the behaviour of the model satisfy all its requirements as shown below, where $\models$ represents "satisfies":
   $Stage1\ model \models REQ$

We make no attempt to specify how the requirements are constructed because it is dependent on the language and toolkit used. Nor are we concerned with how the model is implemented. In step four, if $stage1\ model$ does not satisfy the requirements, it is necessary to confirm that the requirements are correctly constructed. It may be that the behaviour of the model reveals an error in the requirements construction. When the model satisfies all requirements, stage one has produced a $Stage1\ model$ and REQ that will be used in the next two stages.

5.2 Stage Two

The aim of this stage is to detect any interactions between the $Stage1\ model$ and the integration of a single feature. It is not advisable to modify REQ as it would invalidate the results from stage one. We repeat this stage for each feature, resulting in multiple models containing only the integration of a feature with the base model. The integration of the feature requires modification to $Stage1\ model$ to derive $Stage2\ model.$

1. Construct requirements for the feature ($P1$):
   This step is similar to that done in stage one.
2. Model the feature ($F1$): Language dependent.
3. Integrate $F1$ with $Stage1\ model$ ($Stage2\ model$): Language dependent.
4. Verify $Stage2\ model$ for general properties:
   On discovery of an error, resolve the error and repeat this step.When all errors are resolved, progress to the next step.
5. Validate $Stage2\ model$ against REQ and $P1$ requirements:
   First REQ is validated. If REQ is not satisfied, the integration of the feature (including solutions to general property errors from the previous step) has modified the behaviour of the model to an **undesired** level, that it no longer satisfies its original requirements. What is done at this point is left to the model implementors. One solution is to review and modify resolutions to errors in step four and then to repeat that step before progressing to this one. When REQ is satisfied, then validate $P1$. If $P1$ is not satisfied and providing it is correctly constructed, resolve the error and then repeat from step four (including validation of REQ in this step to ensure it is still satisfied). If $P1$ is incorrectly specified, use the validation result to modify $P1$ and repeat this step. The aim of this step is to ensure $Stage2\ model$ satisfies the conjunction of the original requirements and those of the feature's:
   $Stage2\ model \models REQ \wedge P1$

5.3 Stage Three

This is the feature interactions detection stage. This stage allows integration of more than one feature to an existing $Stage2\ model$. Which features are integrated depend on issues such as what services are to be provided or analysis of combination of features. The integration of features requires modification to a $Stage2\ model$ to derive $Stage3\ model$.

1. Integrating additional features to a $Stage2\ model$ ($Stage3\ model$):
   Using one of the $Stage2\ models$ as a starting point, integrate features $F2 \ldots Fn$ to the model to derive $Stage3\ model$. This must include all solutions to general property errors from $Stage2\ model$ for all the features integrated.
2. Verify $Stage3\ model$ for general properties:
   On detection of an error, resolve the error and then repeat this step.
3. Validate $Stage3\ model$ against $REQ \wedge P1 \wedge P2 \wedge Pn$ requirements:
   This step detects any interactions between the integrated features. The features requirement include only those integrated. If $Stage3\ model$ fails to satisfy any requirements, feature interaction has occurred:
   $Stage3\ model \models REQ \wedge P1 \wedge \ldots \wedge Pn$

Feature interactions is deemed to occur only if the behaviour of one feature prevents another feature from satisfying its requirement. Any errors such as deadlock that occur during step two should be resolved using guidelines applicable to the model. When all errors are resolved, it is the validation step that detects interactions. We know that completion of stage two for each feature in isolation satisfies all its requirements and those of the base model. Therefore, the validation step in stage three will detect any interactions, when a requirement is not satisfied. Any solutions applied to resolve feature interactions means returning to step two, where the cycle begins. Modifying any of the requirements at this stage invalidates the previous stages.

6 Applying the Technique Using Promela and Xspin

In this section, we show how each stage of the technique is applied to the TETRA basic bearer services and the CFU and BIC features. All requirements are constructed using linear temporal logic. We completed the information extraction process before starting this stage. It identified channel *ri*, as the one to include in any requirements containing messages. Consequently a mobile's call setup request is valid only when its *pc* process places a corresponding PDU in the channel.

6.1 Stage One - The Base Model

Constructing the base requirements. To construct a requirement, we apply the following steps:

1. An informal text description for each requirement is provided from the information extraction process.

2. The text description is turned into a temporal formula consisting of propositions and temporal operators.
3. The propositions are defined using messages and variables as parameters identified by the information extraction process.

For example, the text wording for a requirement concerned with all mobiles in the group call states that ,“All mobiles must join the group call and remain in the call.” This is translated into the persistence temporal property $\Diamond\,\Box(group\_setup)$. The proposition $group\_setup$ is defined as:

```
#define group_setup (IdleA == false && IdleB == false
                     && IdleC == false)
```

For the property to be satisfied all the variables must have the value false, which indicates they are in a call.

Modelling the TETRA basic bearer services. We first declare all global variables. To use a string reference for each mobile, we use the *#define* keyword. Each mobile is defined as a two character string [7], though in this paper we refer to the single character. We also define the group identity using the same keyword. The status of a mobile and other mobiles it can connect to, are declared as boolean variables and initialised with values. The example lists only declarations for mobile A:

```
#define  GROUP  0          /* Group ID */
#define  AA     100        /* mobile A */
bool     IdleA = true;     /* mobile is idle */
bool     ConnectAA_BB = false,
         ConnectAA_CC = false;
```

We declare the structure of each PDU to have four parameters using the *typedef* keyword. Primitive types are declared using *mtype* keyword. The channels used for communications between the processes are declared using the *chan* keyword:

```
typedef pdu {byte Tsi,  /* id for this mobile */
                  GTsi, /* Group id for the mobiles */
                  DestTsi, /* Tsi for contact mobile */
                  Msg };    /* additional pdu data */

mtype = {tncc_setup_request, ...};

chan AAra = [1] of {mtype, pdu}; /* A’s ra channel */
chan AArd = [1] of {mtype, pdu}; /* A’s rd channel */
chan AAri = [1] of {mtype, pdu}; /* A’s ri channel */
```

All channels are bounded to one. This allows earlier detection of deadlock [8] during verification and validation. After declaring the global variables, we declare the processes and define their behaviour as:

[7] To allow features like Short Number Addressing (SNA) to reduce it to one character.
[8] Caused by processes being unable to place a message in a full channel.

```
proctype SwMI()
{end_IDLE: atomic{...}}     /* SwMI's behaviour */

proctype pc(chan rd,        /* rd channel */
                 ri)        /* ri channel */
{...}                       /* pc's behaviour */

proctype cc(chan ra,        /* ra channel */
                 rd)        /* rd channel */
{...}                       /* cc's behaviour */

proctype ua(chan ra,        /* ra channel */
            byte UATsi;     /* mobile's identity */
            bit  AGTsi;     /* mobile group id */
            byte CanCall_1, /* mobile to call */
            byte CanCall_2) /* mobile to call */
{...}                       /* ua's behaviour */
```

As all processes in the base model do not terminate, we use the *end* keyword to indicate expected end states as shown for the SwMI. Since the SwMI coordinates communications between mobiles, we also use the *atomic* keyword to ensure it executes as many statements as possible without interleaving with the other processes. The behaviour of each mobile is constructed using the same pc, cc and ua processes. These processes are parameterised to identify the channels associated with each process upon instantiation. In addition, the ua process knows the identity of itself, its group and the other mobiles it can nondeterministically make a call setup request to. The ua passes these information to the other processes using the PDU parameters.

We use *init* and *run* keywords to instantiate all processes providing actual parameters for those that require them. As the SwMI process is the coordinating process, it is instantiated before any others to ensure it is ready for communication. For the same reason, the ua process for all mobiles is instantiated after its associated pc and cc processes, to prevent it requesting setup before the others are instantiated. We list only the SwMI and processes for mobile A, mobiles B and C follow the same format to create the environment in figure 3:

```
init{ run SwMI();
      run pc(AArd,AAri);      /* channels rd & ri */
      run cc(AAra,AArd);      /* channels ra & rd */
      run ua(AAra, AA,        /* channel ra & mobile id */
             GROUP, BB, CC);  /* Group Id & can calls */
} /* end of init */
```

Verifying the TETRA model for general properties. We use Xspin version 3.2.4 with the full queue option to block new messages. Use of this option leads to deadlock because of the way the processes are connected. For example, all pc processes fill their ri channels with a call setup request. The ua, cc and pc processes for all mobiles then block waiting for a response to their request. The SwMI non-deterministically removes one of the requests from a mobile's channel *ri* and blocks because it is unable to place a message on the called mobile's channel to inform it of an incoming request. The deadlock is reported by Xspin, which allows the use of the guided simulation to analyse its cause. All errors reported by Xspin were of similar nature and were resolved. Two error free models were implemented at the end of this step:

Tx model - On joining the group mobiles could request and be granted permission to transmit. During hash compact verification, it uses thirty one megabytes of memory.
Non Tx model - Mobiles cannot request transmit permission on joining the group call. They can non-deterministically reject or accept requests to join the group. During exhaustive verification, it uses seven megabytes of memory.

Validating the TETRA model against its requirements. The Tx model failed the persistence property requirement $\Diamond\,\Box\,(group\_setup)$. Its guided simulation revealed that transmission was (unintentionally!) assigned a higher priority than call setup. Thus, when two of the three mobiles joined the group call[9], their transmission activities prevent the SwMI from connecting the third mobile to either of the two. The Non Tx model on the other hand, satisfied all the requirements. Taking this into consideration and that it used considerably less memory than the Tx model, we used it for the next two stages. The alternative solution would be to correct the Tx model by assigning call setup a higher priority than transmission.

6.2 Stage Two - The CFU Feature

In this section, we show the steps used to integrate a feature with the Non Tx model. Though we use the Call Forwarding Unconditional feature as an example, the same procedures are followed for the Barring of Incoming Calls feature. We completed an information extraction process for each feature before starting this stage. It helped to identify the variables and messages required by the feature.

Constructing requirements for the CFU feature. This step follows the same step as for the base model requirements. We found that the feature's requirements should include the following amongst others:

– Activation and deactivation of the feature.
– All communications between the feature and mobiles.
– Global variables modified by the feature.

For example, the requirement, "The CFU informs the calling mobile of acceptance of its request by the called mobile," is translated into the response temporal property $\Box\,(cfu\_divert\_accept \rightarrow \Diamond\, cfu\_accept\_request\_ms)$. Proposition *cfu_divert_accept* defines the conditions that represents the called[10] mobile informing the CFU of acceptance. Proposition *cfu_accept_request_ms* represents the CFU informing the calling mobile of acceptance. They both are defined as:

[9] One accepting the request of the other.
[10] The mobile, the calling mobile was diverted to.

```
#define cfu_divert_accept
            (BBri?[pc_swmi_u_connect,BB,0,AA,255]
             && CfuActive_c == true)
#define cfu_accept_request_ms
            (AAri?[swmi_pc_d_connect,AA,0,BB,255]
             && ConnectAA_BB == true
             && ConnectAA_CC == false
             && CfuActive_c == false
             && IdleA == false && IdleB == false)
```

The response property [11] uses the implication symbol, therefore the only time the property is not satisfied is **if** proposition *cfu_divert_accept* occurs (i.e. the required parameter values occur in the same state) and *cfu_accept_request_ms* does not occur in future states . As we use Promela's conjunction operator (*&&*), a proposition is false if any one of its arguments does not match its required value when the others do in the same state. With this in mind, we intentionally use the minimum number of parameters in the definition for proposition *cfu_divert_accept*, so it has more of a chance of occurring. We also maximise the number of arguments for proposition *cfu_accept_request_ms* that allow it to occur. As the CFU is the only feature in the model, this is straight forward. We include only parameters that relate to the mobiles involved in the feature's task.

Using the parameters in this way contributes greatly towards detecting feature interactions in stage three. Minimising the arguments in the first proposition of the property reduces the likelihood of another feature falsifying it. Maximising the number of arguments in the second increase the possibility of it becoming false due to another feature in stage three, modifying one of its arguments.

When messages are used in propositions definition, we refer to specific channels , PDU primitive types and PDU parameters that are required. Being able to specify the PDU parameters in definitions allow us to ensure a response is a result of a previous action. For example, we know that the messages included in the above propositions are related because the first parameter (which identifies the mobile) and the third (its contact) match when reversed.

Modelling the CFU feature. All features are designed to terminate on completion of their task. It is necessary to introduce additional global variables [12] that specify the status of the feature, its subscriber, the calling and the divert to mobile. To allow flexibility, we use parameters in the definition of all features:

```
proctype cfu(byte SubscriberMs, /* mobile id */
                  CallingMs,    /* calling mobile */
                  DivertMs,     /* mobile to divert to */
                  GroupId,      /* group id */
                  Message)      /* additional pdu data */
{IDLE: atomic{...}}             /* cfu's behaviour */
```

[11] As the TETRA network is distributed, the response property is used for most of the requirements in the first two stages.

[12] Identified from the information extraction process for the feature.

We consider the CFU to be part of the SwMI and for this reason, it also uses the *atomic* keyword. Using the CallingMS and DivertMs parameters, the feature is able to dynamically decide which channel *ri* to use for communication with the mobiles.

Integrating the CFU with the base model. To integrate the CFU into the base model from stage one, we modify the SwMI process by using the *do*. . .*od* construct to instantiate the feature:

```
do /* activate features */
:: (CfuActive_c == false &&   /* not active for mobile */
   CfuAlreadyRan_c == false)-> /* not ran for C */
   CfuActive_c = true ;       /* to be activated */
   CfuAlreadyRan_c = true ;   /* cfu will be ran  */
   FeatureCounter_c++ ;       /* feature activated */
   run cfu(CalledTsi,         /* subscriber mobile */
           CallingTsi,        /* Calling mobile */
           BB,                /* mobile to divert to */
           GTsi,              /* group id */
           Msg);              /* additional message */
:: /* termination of feature activation */
   (FeatureCounter_c == 1) -> /* total num. features */
   CfuAlreadyRan_c = false ;  /* cfu can run again */
   FeatureCounter_c = 0 ;     /* all features can run */
   goto end_IDLE ;           /* Swmi handle other calls */
od; /* activate features */
```

To run the CFU, the SwMI checks that the feature is not currently active and that it has not already run. If these conditions are met, the SwMI sets the status of the feature as active and marks the feature as already ran. The feature counter is then incremented, to show a feature has been activated. The CFU is then instantiated with parameters. To break out of the construct the SwMI uses the termination sequence. The feature counter must equal the total number of features activated for the subscriber. The feature is then marked as available to be ran. The feature counter is reset to zero and the SwMI is able to handle other calls. The CFU is responsible for reseting its status, to prevent the SwMI from instantiating two copies of the same feature. The SwMI blocks until the active CFU terminates before instantiating another copy of the feature. Providing actual parameters for the CFU, models the behaviour of the supplementary services sub-entity discussed in section 3.2.

Verifying the integrated model for general properties. Since both the CFU feature and the SwMI access channel *ri* to communicate with the mobiles, violation of an assert statement during verification revealed a race condition occurs with the SwMI consuming messages meant for the CFU and vice versa. We resolved this error by replacing the assert statement with additional code so that both SwMI and CFU, on retrieving a message, check whether the message is intended for itself. If this is not the case, the process puts the message back in the channel and blocks until it is removed by another.

Using the hash compact search, verification of the model used fourteen megabytes of memory.

Validating the base model and CFU requirements. This step revealed that propositions using messages parameters are influenced by removal of messages from channels. For example, the response property for the CFU in its original form was not satisfied. This is because the *cfu_accept_request_ms* proposition relied on the argument *CfuActive_c* equalling true (meaning the CFU feature is active). Analysis of its guided simulation revealed that when the message is placed in the channel by the CFU, the proposition is satisfied. However, by the time the calling mobile removes the message, the CFU is no longer active causing the requirement not to be satisfied. This is a case, where the acceptable behaviour of the model resulted in the requirement being modified to the form shown on page 11. Simply changing *CfuActive_c* to equal false resolved the error.

All base model requirements were satisfied provided they refer to mobiles that do not subscribe to the CFU feature [13]. This confirms that the requirements from stage one need to be included in the second and third stages, in order to ensure mobiles that do not subscribe to features are unaffected by feature integration. It is sufficient to validate requirements one at a time, as failure to satisfy an individual requirement will not satisfy their conjunction.

6.3 Stage Three - Feature Interactions Detection

Integrating the BIC with the CFU and base model. To answer the uncertainty raised in section 2, we integrated the BIC to the CFU's $Stage2\ model$. As both the CFU and BIC are activated by the same message in the SwMI, we added the code required for the SwMI to instantiate the BIC in the same *do*. . .*od* construct used for the CFU. During instantiation, the BIC is supplied with actual parameters. In the listing below, modifications to the termination sequence is required to cater for integration of the BIC:

```
do
:: /* instantiate CFU  */
:: /* instantiate BIC  */
:: /* termination of feature activation */
   (FeatureCounter_c == 2) ->  /* total num. features */
    CfuAlreadyRan_c = false ;  /* cfu can run again */
    BicAlreadyRan_c = false ;  /* bic can run again */
    FeatureCounter_c = 0 ;     /* all features can run */
    goto end_IDLE ;        /* SwMI handles other calls */
od ; /* activate features */
```

All solutions required to resolve deadlock errors in stage two for the BIC were also included in the SwMI and pc processes. An interesting observation was that the BIC solutions were found to be a subset of the solutions required for the CFU. This is because when the features have carried out their specific function (e.g. the BIC screens the incoming request), they behave identically.

[13] This is because the CFU takes over parts of the SwMI's operations.

Verifying the model for general properties. The first deadlock error revealed that the CFU does forward a mobile that is on the subscriber's barred list to another, whilst the BIC bars the same mobile from contacting the subscriber. As far as we are concerned, both features are behaving as expected and this is not a case of feature interactions, so we continue with resolving all verification errors.

The final hash compact verification required two hundred and sixty nine megabytes of memory. Xspin also raised an "out of memory" message [14], indicating the state space exploration was not complete. Though the number of errors found was zero, we cannot conclude that the model is error free. However, we can conclude no errors were found in the state space explored. With this in mind, we progress to the next step.

Validating the base model, CFU and BIC requirement. Though the state space exploration was not completed in the previous step, we can still continue with the validation step. The aim is to see if the behaviour of the model violates any requirement, before memory constraints terminate the exploration.

We found two cases of feature interaction that did not exist in stage two for either feature in isolation.

– Though the BIC satisfied its requirements, its behaviour prevented the CFU from satisfying the requirement discussed on page 11. Feature interactions occurs because the BIC sets the status of the barred mobile as inactive, when it is rejected. However, the CFU continues to connect the barred mobile with the mobile it is diverted to. The CFU's requirement expects the status of the barred mobile to be active when connection is achieved. The behaviour of the BIC, causes the argument *IdleA == false* in proposition *cfu_accept_request_ms* to be false. As the proposition *cfu_divert_accept* has occurred, the requirement is not satisfied.
– The behaviour of the BIC in setting the status of the barred mobile as inactive also prevented the original requirement from stage one discussed on page 9 from being satisfied.

Discovering the interactions used only six megabytes of memory. This is because violation of the requirement was detected early during validation, therefore requiring less memory to store the generated state space.

7 Conclusion

We introduced a three stage technique that detects feature interactions. Each stage consists of iterative steps geared towards achieving specific objectives. The first stage produces a base model that satisfies its requirements. Stage two builds on the base model by integrating a single feature. Isolating the feature from others in this stage, allows correct construction and validation of its requirements. Successful completion of stage two results in a modified model that satisfies its original requirements and those of the

[14] Of all feature integrations the CFU and BIC prove to be the most complex and resource demanding to verify.

integrated feature. This stage is performed for all features. The final stage allows multiple features to be integrated to a model from stage two. Feature interactions is detected if the modified model does not satisfy any of its original requirements and those of the features integrated.

We demonstrated the effectiveness of the technique using Promela to model the TETRA network and two features. The Xspin toolkit was used to verify and validate the model in every stage. The most important benefit in using the Promela and Xspin combination is the ability to validate models against requirements using linear temporal logic. Since validation forms the heart of our technique, using Promela and Xspin in this manner allows proof of concept.

Acknowledgement

The authors would like to thank Neil Evans, Huma Lodhi and Helen Treharne for their comments on this paper. We also thank Motorola's Paul Baker and Clive Jervis for discussions about the TETRA network.

References

[Eura] European Telecommunications Standards Institute. *Radio Equipment and Systems (RES); Trans-European Trunked Radio (TETRA); Voice plus Data (V+D); Part 10: Supplementary Services Stage 1; Part 10-04: Call Diversion.* ETS 300 392-10-04.

[Eurb] European Telecommunications Standards Institute. *Radio Equipment and Systems (RES); Trans-European Trunked Radio (TETRA); Voice plus Data (V+D); Part 11: Supplementary Services (SS) Stage 2; Part 11-19: Barring of Incoming Calls (BIC).* ETS 300 392-11-19.

[Hol90] Gerard J. Holzmann. *Design and validation of computer protocols.* Prentice Hall, 1990. ISBN 0-13-539925-4.

[JP98] M. Jackson and P.Zave. Distributed Feature Composition: A Virtual Architecture for Telecommunications Services. *IEEE Transactions on Software Engineering,* 24(10):831–847, October 1998.

[KK98] D.O. Keck and P.J. Kuehn. The Feature and Service Interaction Problem in Telecommunications Systems:A Survey. *IEEE Transactions on Software Engineering,* 24(10):779–796, October 1998.

[LL94] F. Joe Lin and Yow-Jian Lin. A Building Block Approach to Detecting and Resolving Feature Interaction. In W. Bouma and H. Velthuijsen, editors, *Feature Interactions in Telecommunication Systems,* chapter 6, pages 86–119. IOS Press, 1994.

[MP91] Zohar Manna and Amir Pnueli. *The Temporal logic of Reactive and Concurrent Systems Specification.* Springer-Verlag, 1991. ISBN 0-387-97664-7 (v. 1).

[SDL] *Annex F to Recommendation Z.100, (Formal Definition of SDL 92), Dynamic Semantics.*

[TM98] S. Tsang and E.H. Magill. Learning To Detect and Avoid Run-Time Feature Interactions in Intelligent Networks. *IEEE Transactions on Software Engineering,* 24(10):818–830, October 1998.

[TR95] ETSI TC-RES. *Radio Equipement and Systems (RES); Trans-European Trunked Radio (TETRA); Voice plus Data (V+D) Part 2: Air Interface (AI).* European Telecommunications Standard, 1995. ETS 300 392-2.