Modelling and Analysing User Views of Telecommunications Services

Muffy Thomas
Department of Computing Science
University of Glasgow[*]

Abstract

User views of calls are modelled by behaviour trees, which are synchronised to form a network of users. High level presentations of the models are given using process algebra and an explicit theory of features, including precedences. These precedences abstractly encapsulate the possible state spaces which result from different combinations of features.

The high level presentation supports incremental development of features and testing and experimentation through animation. Interactions which are not detected during the experimentation phase may be found through static analysis of the high level presentation, through dynamic analysis of the underlying low level transition system, and through verification of temporal properties through model-checking. In each case, interactions are resolved through manipulation of the feature precedences.

keywords

formal specification and modelling; analysis and reasoning techniques; feature interaction detection and resolution.

1 Introduction

Telecommunications services are increasingly pervasive; it is therefore important that services deliver the expected behaviour. On the other hand, as service providers develop more and more of their services in software, and a deregulated market encourages multiple providers, the potential for interactions, or interworking inconsistencies, between and within services grows expontentially. The detection and resolution of these interactions, known as *feature interactions*, has been the focus of research in telecommunications and telephony systems for the past few years (e.g., see [4, 5, 6]). In this paper we develop off-line analysis and resolution techniques focussing on interactions from users' viewpoints; the results might also inform some on-line or hybrid techniques.

We describe an approach to specification and modelling which allows the systematic detection and resolution of certain classes of feature interactions. The approach involves specifying properties in a temporal logic and modelling calls by behaviour trees. While the model reflects some aspects of the operational world (i.e. the implementation software) it mainly reflects a high-level task analysis from users' viewpoints. It provides a testbed for studying the phenomena and for developing

[*]Supported by a BT Laboratories Fellowship

features, as well as a concrete model against which the satisfaction of more abstract properties can verified. An explicit model also avoids the frame problem – particularly acute in distributed systems.

Through analysis, we can uncover and resolve not only predictable interactions, as encapsulated by specific abstract properties, but also some unpredicted ones through systematic analysis of the model. Since the state spaces involved are enormous, for any number of features and users, we employ a high level presentation using a message passing process algebra. Possibly unique to this presentation is an explicit theory of features which guides the generation of the trees. Features are both static and dynamic entities (we call the latter *modes*) and the theory includes precedences – complex functions which depend on call states. In effect, we have incorporated a form of negotiation between features based upon expected precedences for particular combinations of calls and their respective states. That is, a choice between two alternative behaviours is made based upon the (feature) context in which the respective actions are being offered. In the model, certain classes of interactions manifest themselves as additional, unexpected non-determinism; when this is detected, it can be resolved by manipulating the feature precedence relations.

In the following section, we briefly describe the background and context of feature interaction problems. Following that, an overview of the approach is given in Sections 3 and 3.1. Details of user processes and feature theories are given in Section 4, modelling an entire system and particular scenarios is discussed in Section 5. Analysis techniques and results are discussed in Section 7 and we conclude in Section 8.

2 Background

Features of a telecommunications service are said to *interact* when one changes the functionality of another(s) (see [7] for a categorisation of feature interactions). As the complexity of the feature interaction problem has unfolded, it has become clear that many interactions arise from invalid assumptions about features, services, and network environment, throughout all phases of software development. Thus the need for specifying and/or designing services more rigorously has become universally recognised. The scale and speed of system evolution strongly motivates the use of automated analysis tools.

A variety of formal description techniques have been used for both system models and abstract properties, including finite state machines and their extensions (e.g. SDL and Promela), labelled transition systems, process algebras (e.g. LOTOS), state based notations (e.g. Z and Object-Z) classical, temporal and non-monotonic logics [3, 12, 17]. The formal approach is typically stated (e.g. in [3, 17]) as: given descriptions of features F_1, F_2, network N and properties Φ_1 and Φ_2, when $N \oplus F_1 \models \Phi_1$ and $N \oplus F_2 \models \Phi_2$, then we expect $N \oplus F_1 \oplus F_2 \models \Phi_2 \wedge \Phi_2$; $\oplus$ is a "combination" operator. If $N \oplus F_1 \oplus F_2 \not\models \Phi_2 \wedge \Phi_2$, then there is an interaction. In other words, we expect the behaviour of one feature to be preserved in the presence of another feature. But, there are several problems with this approach. First, it has to be qualified carefully. Namely, failure to satisfy such a meta-property does not necessarily indicate an undesirable interaction: some features must necessarily alter, or override some aspects of other features or the network (e.g. 999 calls cannot be terminated by the callers, call waiting alters properties of the busy state).

Second, the definition of the $\oplus$ operator is usually left undefined (thus raising questions such as to why a formal notation has been used at all!). Velthuijsen [17] raises several important concerns about the possible nature of this operator; in particular, he notes the need to incorporate some aspects of the non-monotonic nature of network extensions. Our model makes the behaviour of this operator explicit.

3 The Model

The high level model is motivated by the need to design features *independently* of each other, and to reuse existing behaviour as much as possible. For example, if a feature only affects originating call behaviour, then when designing that feature, one should not have to redefine the terminating behaviour, in the context of that feature. The model we have developed is most naturally expressed at a high level in a message passing process algebra with expressive data theories and guarded processes. Therefore LOTOS [11] was adopted as the presentation language. While this language does have some drawbacks, it is well-supported with toolsets e.g. the LOLA/TOPO toolset [15] for simulation and abstract property checking and the CAESAR toolset [9] for model-checking.

3.1 An Overview

A telephony system may be regarded as a distributed system, where the primary components are user processes. The salient aspects of a user's view of such a system are its internal state, its active features, and possibly the state and active features of other users. Thus, we must incorporate some "awareness" of the global state.

At the most abstract level, the model consists of user processes and a network manager. The network manager is a process which can receive and transmit information about users, from and to users. The current model is designed specifically for two-party calls (though it could be be extended to multi-party calls). The model is reasonably abstract, in that we have chosen to identify lines, users, and numbers. (We regard this particular model as a prototype, designed to be only realistic enough to demonstrate the utility of this approach.)

User processes move through various internal states, engaging in actions according to the state. Concurrency in the system is modelled through synchronisation; thus some actions are required to synchronise with others. An entire system consists of the user process(es) and network process in parallel, synchronising on particular actions, as shown in Figure 1. Synchronisation[1] is typically required between two (or more) user processes (e.g. a connection is established) or between one user and the network manager (e.g. receive or transmit information about a user state). The latter actions are unobservable at the system level.

It is helpful, when describing the model, to distinguish between two kinds of actions:

- "user initiated" actions such as pick up the handset, replace the handset, dial a number, offer speech etc. These actions represent physical actions which are initiated by a user and may involve synchronisation with another user. Synchronising actions between users x,y, .. x have the form "action!x!y..!z".
- "network initiated" actions such as busy tone, line unobtainable. These events are typically experienced by a user, but they are not initiated by a user, rather they are a consequence of the global state of the network.

It is important to note that a single "physical" action may have several representations in the model; the context in which the action takes place determines the representation. For example, a user replacing the handset in order to disconnect a line, i.e. the user is the originating caller, is represented by the (user initiated) synchronising action "disconnect!x!y", whereas a user replacing the handset when

[1] the operator "|[. . .]|" denotes synchronisation on the actions in the list [. . .])

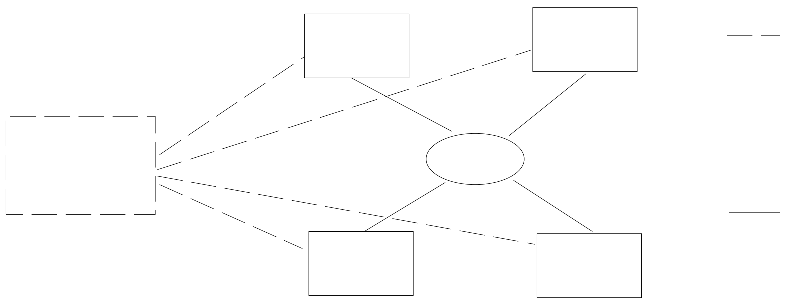

Figure 1: A Four User System Model

the line has been disconnected, i.e. the user is the terminating party, is represented by the action "on".

As an example, the transition system for basic call behaviour is given in Figure 2. User initiated actions are given in times font, network initiated actions are given in courier font. In the synchronising actions, "id" refers to the user process identification, and "pid" to refers to the partner's process identification (n.b. all synchronisations in basic call behaviour are between two users). For example, "disconnect!pid!id" denotes the action of disconnecting the call from the partner of a user to a user, thus the user is the terminating party in the call and the partner is the originating party.

3.2 Modelling Static and Dynamic Aspects of Features

The high level presentation allows for state abstraction in the following way. Each user process is deemed to be operating in a set *modes* and each mode determines a set of possible behaviours for a given state. That is, a mode is an abstraction of an active feature and we use it to determine possible transitions, or actions, for a given state. For example, Basic call behaviour is considered as a mode. In Figure 2, the modes are not given explicitly, but one should assume that each action is only offered when the user process is in basic call mode.

In our model, the set of *all features* to which a user process subscribes (some of which may not be active), is called its *service*. Each user process is parameterised by its modes, its service, and its current call "partner" (i.e. the other party in two party calls). Modes are *dynamic* while a service is *static*. It is important to note that a mode does not not necessarily mean that a feature influences current behaviour (i.e. it does not exactly correspond to an active feature), it only may effect future behaviour. Typically, in an initial, idle state, the modes of a user process are the same as its service – all features are (potentially) active. The modes evolve as the call evolves; usually reverting back to the service upon termination of the call.

This evolution of modes during a call allows features to be built up incrementally, reusing much of the previously specified behaviour. For example, a user process in any sort of diversion mode (e.g. divert when busy, divert on no reply) evolves into a process in basic call mode when it is the originator of a call. So, when designing new features, new user states are only added when new behaviour is required. In other words, modes allow us us to abstractly group together states into (super)states. For example, the basic-call-calling-state, divert-busy-calling-state and divert-no-reply-calling-state are all encapsulated in the same calling (super)state.

Features are not discrete, but are *ordered* according to partial orderings. These orderings make explicit how potential interactions may resolved, and allow experimentation with different orderings.

There are two types of ordering: *intra-user* orderings and *inter-user* orderings,

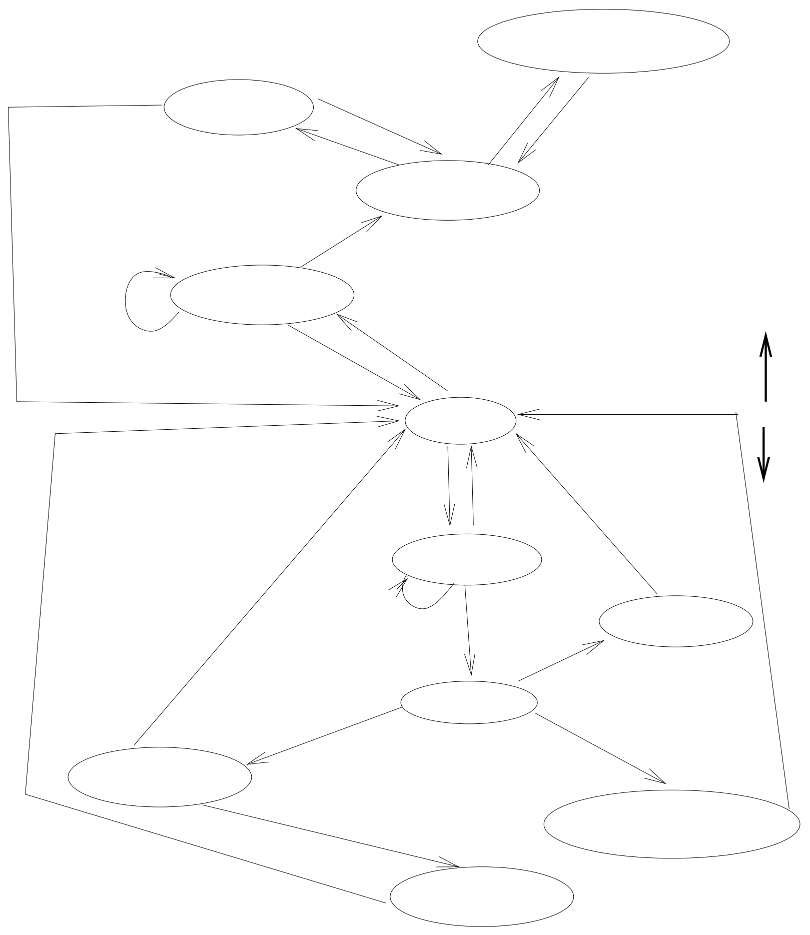

Figure 2: Basic Call

both of which depend on the state(s) of the user processes involved. The former define priorities between features for a given user. For example, if a user is behaving in both a divert always mode and a divert busy mode, then the intra-user ordering, for the idle state, may determine which feature has precedence the user is called. (Of course, the relations need not be total, and so the model may not be deterministic.) The latter define priorities between features involving different users, that is, the behaviour of one user may be determined by another users' state, modes and services. This means that some information about user modes and services, as well as state, must be received and transmitted by the network process.

We will return feature precedences and mode evolution in section 4.1, in the next section we give an overview of a user process.

4 User Processes

Each feature is designed separately as a (sub) process; the overall structure of a user process is a *choice* (i.e. logical disjunction) between the processes associated with the features, as shown in figure 3. "[]"is the LOTOS operator for choice.

Essentially, each feature (sub) process offers a number of alternative actions, depending on the the user's current state, mode, etc., and possibly the mode and state of its partner. The alternative are *guarded* by the appropriate predicates which

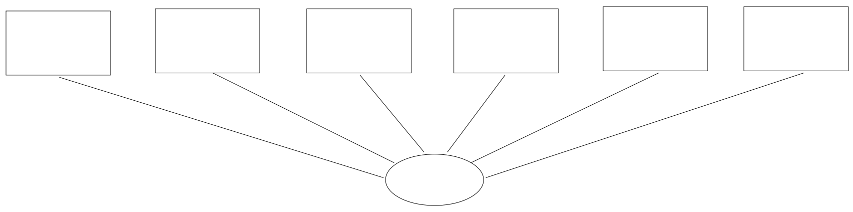

Figure 3: A User Process

are defined in the (data) theories of states and modes. These include *state predicates* such as "idle(u)" (user u is in idle state) and *mode predicates* such as "bcmode(u)" (user u is in basic call mode).

The choice operator is employed, both for combining alternatives within a feature behaviour, and for combining features.

Each user process is parameterised by the current state, current modes, and current partner (collected together explicitly in a data type with sort of interest "user"). Ignoring the formal and actual events in process declarations and calls, and assuming some unary functions "f1", "f2" on sort "user" which alter the state and/or modes, such a process has the form :

```
process call(u:user) :=
     [statepred1(u) and modepred1(u)] → action1; call(f1(u))
[]   [statepred2(u) and modepred2(u)] → action2; call(f2(u))
[]   ...
```

While here, for simplicity, one (observable) action is followed by a recursive call; in general, more complex sequences or choices (involving unobservable actions), are offered. For example, the process corresponding to basic "plain old telephony" behaviour (Figure 2), is given by:

```
BasicCall(u:user)[..] :=
     [idle(u) and BCmode(u)] →
          off; status!write!id(u)!busy; Call[..](user(id(u),candial,mode(u),partner(u)))
          []
          connect?partner:id!id(u); Call[..](user(id(u),Talert,mode(u),partner(u)))
[]   [candial(u) and BCmode(u)] →
          dial?y:id; Call[..](user(id(u),calling,mode(u),y)))
          []
          dialtone; Call[..](user(id(u),candial,mode(u),partner(p)))
          []
          on; Call[..](user(id(u),idle,mode(u),partner(p)))
[]   [idle(u) and BCmode(u)] → ...
[]   [calling(u) and BCmode(u)] → ...
[]   ...
```

Here, "user" is the constructor for sort "user": its first operand is a user identifier, the second operand is a state (e.g. candial, calling, etc.), the third operand is a mode, and the final operand is the partner. "id", "mode", and "partner" are just a selectors. Note that the mode of the user does not change throughout the basic call. A simple example of a mode change can be illustrated by a diversion feature. In such a feature, an originating call in an idle state can evolve into a call in the same state in basic call mode. For example, assuming that "DAltoBCmode(u)" replaces the divert always mode of "u" with the basic call mode, the divert always feature is given by:

Figure 4: Some feature precedences

```
DivertAlways(u:user)[..] :=
    [idle(u) and DAlmode(u)] →
        Call[..](user(id(u),candial,DALtoBCmode(u),partner(u)))
[]  ...
```

When designing a feature, it is important to specify the new behaviour which arises when a user is both an originating party, as well as a terminating party, and to cover all possible *reachable* state and mode combinations. (Typically, one needs only to consider those reachable from the idle state.) This means that for a given new feature (and hence mode) one may need to consider the consequences of satisfying new guards associated with the mode, as well as new consequences of existing guards, e.g.

```
FeatureX(u:user)[..] :=
    [idle(u) and FXmode(u)] → ...
[]  [idle(u) and BCmode(u)] → ...
```

Non-determinism arises from unguarded choices (e.g. from the idle state, in basic call mode), or, more subtly, from *overlapping* guards. State predicates are disjoint, e.g. $\nexists$u.idle(u)$\equiv$ candial(u), but a call may be in more than one mode (see below), thus mode predicates may overlap.

4.1 Mode Evolution and Precedences

In the LOTOS model, modes are represented concretely by sets of feature constants, each of which denotes a particular (operational) feature. Mode predicates do not refer simply to the presence, or absence, of a feature constant in a set, but are true when the feature is also maximal in the intra-user feature precedence with respect to the state of the given process. This rich theory of features allows one to model quite subtle feature behaviour, and in particular, their behaviour with respect to each other.

As a mode evolves during a call, the corresponding feature constant is replaced in the feature set. A common mode evolution is to basic call (BC). For example, assume that when a user in divert always mode (DAl) initiates a call, then that mode evolves to BC (i.e. the diversion is only "active" when the user is being called). If a user is behaving in both a divert always (DAl), a divert busy (DBu), and a basic call mode, then is it has modes {DAl, DBu, BC}. If in the intra-user ordering, for the calling state, only DAl is maximal, then the overall mode is DAl. When the user initiates a call, then the modes will evolve to {BC, DBu, BC}, and if DBu is maximal, then the new overall mode will be DBu. (But note that any reasonable specification of the DBu feature would also stipulate that DBu evolves to BC when the user initiates the call.)

Figure 4 shows a portion of an example intrafeature precedence. Divert always has precedence over the other diversions (e.g. divert busy (DBu) and divert no reply (DNR). Call waiting (CW) has precedence over divert busy, for the idle state, but the precedence is reversed, for the current speech state – a speech state from Call Waiting (see Section 6 for a description). This means that during an on-going two-party call, the first incoming call will activate the call-waiting alert signal, whereas the second incoming call will be diverted to another user. With these feature precedences, it is sensible to offer both call-waiting and a diversion in the same service. Feature precedences are crucial, as they indirectly control behaviour of the call processes. Moreover, if for a given user state and feature list, two mode predicates are true, then we *may* have an undesirable source of non-determinism (i.e. a feature *interaction*).

5 Overall System

The overall system is defined by the parallel, synchronised composition of (instantiated) user call processes, and an (appropriately instantiated) network process. Synchronisation is pairwise between user processes, on the actions which they share, and between all the users and the network. Both single user single component (SUSC) and multiple user multiple component (MUMC) [7] scenarios can be modelled by instantiating the appropriate user processes.

A user process is instantiated by providing an actual "user" and renaming the (observable) actions. That is, the actions in the (generic) process (e.g. "dial") are renamed appropriately, e.g. "dial1" for user 1, "dial" for user2, etc..

The network manager process provides access to the external view of each user. It does so by offering to read and write information about the externally observable states and modes of calls through an unobservable structured event "status" on which any call process can synchronise.

6 Example Features

The (high level) model currently consists of about 3,500 lines of LOTOS and includes a variety of features such as

- Divert Always – all calls to the subscriber are diverted to another user.
- Divert on Busy – all calls to the subscriber are diverted to another user, when the user is not idle.
- Divert on No Reply – all calls to the subscriber are diverted to another user after a pre-determined number (unanswered) rings.
- Dial Barring – the subscriber is not permitted to call numbers in a given *barred* list by *dialling* them directly (this does not preclude a connection between the user and a user on the barred list, which may be possible via a diversion of another user).
- Call Barring – the subscriber is not permitted to be connected to users in a given *barred* list, *ever* (this precludes a connection between the user and a user on the barred list, via a diversion of another user).
- Call Waiting – the subscriber can set up and toggle between two calls.

Since this last feature is significantly more complex than the others, a transition system, with some synchronising actions, is given in Figure 5. This feature is very subtle and given the scope of this paper we can only give a flavour of the behaviour. It is based on British Telecom's call waiting, as described in [1]. In Figure 5, only the new behaviour is included, that is that which results from a user being in call waiting mode, or users in basic call mode which are participating in a call waiting call. For simplicity, no network initiated events are included. Informally, after a first call is established, and a second caller attempts to connect to the subscriber, a special alert tone is issued to the subscriber (and second caller) when the second (incoming) call is received. The subscriber may respond by depressing the R0 R1, or R2 buttons. R0 rejects the second call, and disables any further special alerts (until termination of the first call). R1 disconnects the first call and connects the second call (with no possibility of re-connecting the first call), whereas R2 puts the first call on hold and connects to the second call. In the latter case, subsequent R2 actions allow the subscriber to toggle between the two calls; a subsequent R1 action disconnects the current and connects the holding call. If the subscriber is disconnected from the current call, then the subscriber is rung back and may subsequently become connected to the holding call.

Given this informal description a (subscribing) user may now, in effect, have 2 "partners" in a call – the current "active" partner and the partner "on hold". To model this, we retain the notion of a partner in the user state, (and as before, refer to the partner by the formal parameter pid), but also parameterise the subscriber's call waiting speech state by the "holding" partner, and refer to the holding partner by the formal parameter hid. The subscribing user will always "know" its active and holding partner, but its active and holding partners will only "know" their partners (i.e. the subscriber). As before, in the diagram, we use "id" to denote the user's i.d.

Events may be 2-way, or 3-way synchnronised, the level of synchronisation being indicated by the number of data offers. In the case of 3-way synchronisations, we use the convention that the subscriber is given first, followed by the active partner, followed by the holding partner. For example, imagine that user 1 and 2 are in speech, with 1 the originator and in CW mode, and user 2 in BC mode. User 1 is in Ospeech, and user 2 is in Tspeech. If user 3 (in BC mode, say) attempts to call user 1, then user 3 will be in state calling and can offer the event "alert?x?y!3" (this event was not previously offered as part of Basic Call). User 1 can offer "alert!1!2?x" and move to state "CW sub Speech 2"; user 2 can offer "alert!1!2?x" to move to state "Current Speech". Thus they can all synchronise on "alert!1!2!3" (i.e. the ongoing call from user 1 to 2 receives an alert that there is an incoming call from user 3). As before, the mode and partners of the user process are not represented in the graphical representation. To overcome this, when structured events require substituting or swapping data offers, we use the annotation ("h for p") to denote the former, and ("h&p") to denote the latter. Also, an event may change the user mode to BC; this is denoted by action (mode BC). To reduce the number of arrows, some transitions have multiple labels, separated by a comma, the choice between single and multiple label has no significance, but is dictated by the chosen layout.

7 Interaction Analysis

The relationships between the abstract properties, the high level presentation, and the low level model, and the kinds of interaction analysis which can be performed, are summarised by Figure 6. These include analysis of the logical properties alone, analysis of the properties with respect to a particular model, analysis of both the high level and low level presentations of the model for generic properties, and testing through symbolic simulation. The solid arrows indicate inputs to each kind of analysis, the dashed arrows indicate the feedback to the features theory which may result.

The abstract, logical properties formalise application-specific (mainly temporal)

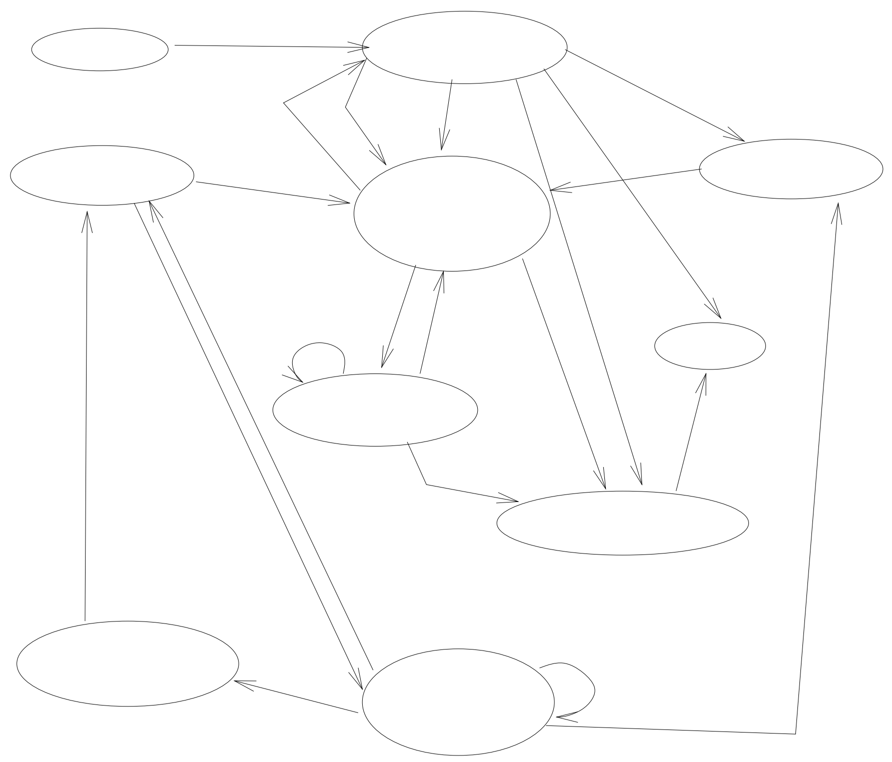

Figure 5: Call Waiting

properties such as "if user n rings user m when user m is engaged, and user m has call forwarding to user p, then user n will be connected to user p". Interaction analysis in this context involves checking conjoined properties for consistency, satisfiability, and sufficient completeness (it is unlikely that we would have, or even desire, complete axiomatisations). Consistency is perhaps the most intuitive concept here, as many interactions will be captured by inconsistent requirements. We note that automating proofs of meta-theoretic properties is very difficult and we have not pursued it.

Analysis of the low level model involves checking for generic properties such as deadlock, livelock, reachability, and unexpected non-determinism. These techniques, and how they may indicate undesirable interactions, are well known from protocol analysis.

Early on, we found that a crucial form of analysis is to run simulations, or tests, on the system in a variety of scenarios i.e. permutations of users and their services. We used the LOLA (*LO*TOS *La*boratory) tool from Madrid [15] extensively, both for animation of tests, and the validation of some properties through abstract testing. The value of these tests cannot be underrated and we discovered numerous small errors this way. When confidence in the model was gained, we moved on to property checking and static analysis, which are described in more detail below.

7.1 Model Checking Temporal Properties

While a number of logics are appropriate for expressing temporal properties, we found the modal μ-calculus[10, 16] a natural one which is also supported by the CAESAR model-checking evaluator [9]. Briefly, the logic includes the usual propositional connectives as well as modal operators. These are $\langle k \rangle$ and $[k]$, expressing existential quantification and universal quantification respectively; with μ and ν

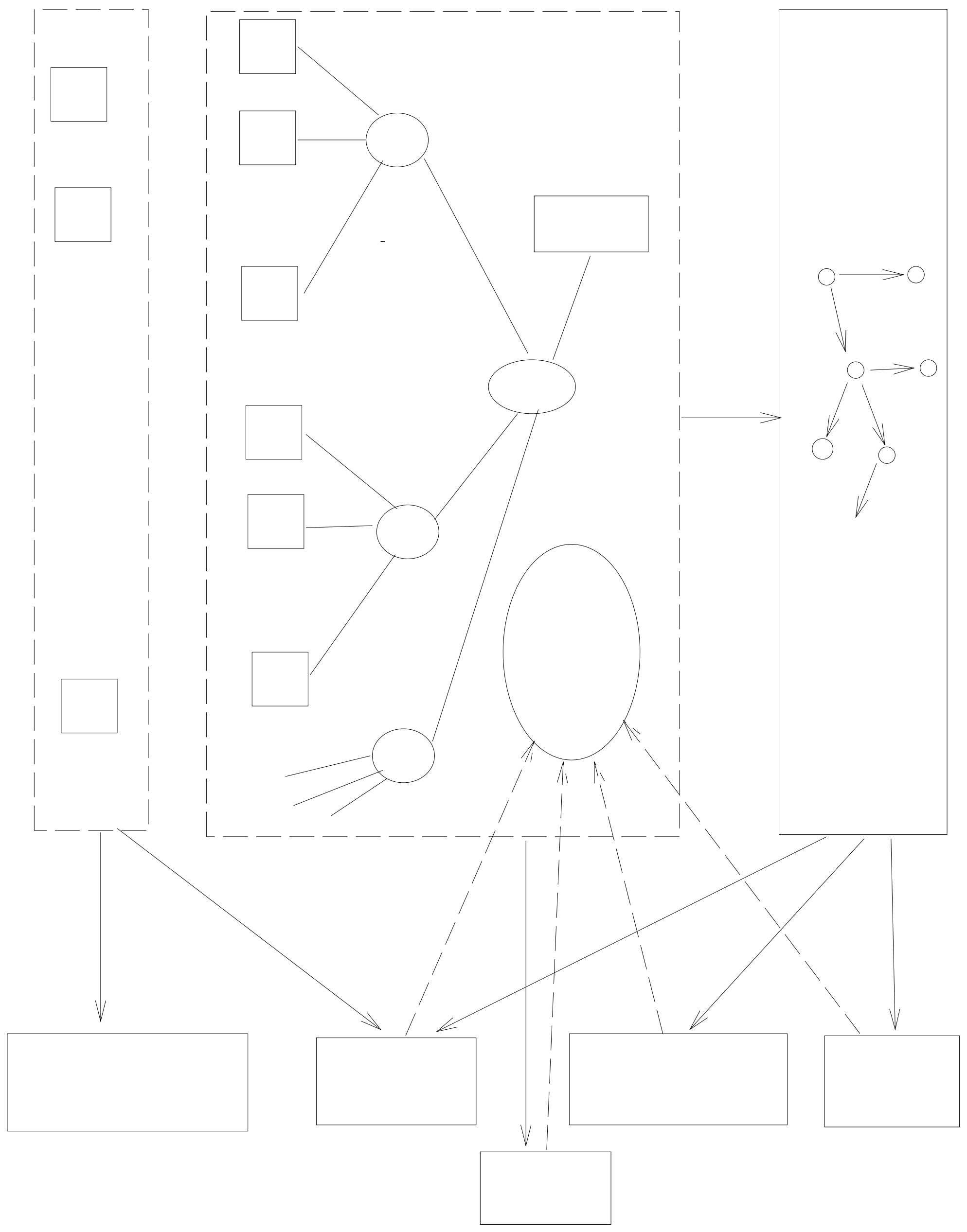
Figure 6: Interaction Analysis

operators denoting the least and greatest fixed point operators, respectively.

As an example, consider a temporal property associated with Basic Call: "after a user calls him/herself, a busytone is issued". We would express this property with respect to a particular user: $\langle off1\rangle\langle dial1!1\rangle\langle busytone1\rangle True$[2]. The evaluator quickly shows that the formula is satisfied, for a scenario where user 1 is in the idle state, in basic call mode. Now consider the case where user 1 is in idle state, in divert always mode, with user 3 as the diversion. In this case, the formula fails to be satisfied – the property has become invalid in the presence of another feature. (Of course this is *desirable* interaction!) A closer examination of the failure (i.e. the "witness" transition system up to the point of failure) can lead to further insights – it is the "busytone1" event which is not possible. In order to find out which other event(s) might be possible, we can try to satisfy formula $\langle off1\rangle\langle dial1!1\rangle\langle *\rangle True$, where where $*$ denotes the wildcard action. This property is satisfied with the "witness" transition system matching the wildcard with event "connect!1!3"[3].

An example of a global property is "it is always the case that if you can lift the handset, then you can't replace it". This can be expressed by the fixed point property $\nu X\ .((\langle on1\rangle True \Rightarrow [off1]False) \wedge [*]X)$. This property was proved, for a single call process, after consideration of 26,170 states and 54,044 transitions, taking 19.7 hours (elapsed time)! This clearly indicates that model-checking in the CAESAR environment is not efficient; moreover, while one can minimilise the transition system with respect to weak bisimulation, sometimes the whole system cannot be generated. So while our initial experience with this toolkit was very promising, it did not scale up to the requirements of the prototype model; a new release of the toolkit may overcome some of these obstacles in the near future. An alternative future direction may be to "compile" the LOTOS into a language such as Promela and use a large-scale model-checker such as SPIN [13].

7.2 Static Analysis of the Model

This analysis is based upon the concept of *interaction* as *non-determinism*. While of course there is, by definition, non-determinism – both in the distributed nature of the system, and within each call process, unexpected non-determinism may indicate a feature interaction.

7.3 Overlapping Guards

Each (observable) event offered by a call process is guarded; thus an overlap *may* may introduce non-determinism. Analysis of overlapping guards consists of finding solutions to conjunctions of guards (e.g. through unification and narrowing), and then considering the consequences of the generated solutions.

As we would expect, we find no such solutions for pairs of guards of the basic call process. For example, there is no user state u such that both "idle(u) and BCmode(u)" and "candial(u) and BCmode(u)" holds. But now consider a possible overlap between the basic call and divert always processes. For example, in the basic call process, an originating call, in basic call mode and calling state, offers a number of choices which depend on the status of the dialled party (i.e. whether it is idle, busy, or unobtainable). These three possibilities are expressed by:

[calling(u) and BCmode(u)] $\rightarrow$

[2]If we have not minimised the transition system with respect to weak bisimulation, then we have to insert unobservable actions, i.e. $\langle i\rangle$'s corresponding to the hidden status events.

[3]Strictly speaking since LOTOS does not allow synchronisation over *structured* events – events with data offers – have to be converted into unstructured events in the actual model, e.g. to the connect13. This unfortunate restriction will be overcome in the next version of LOTOS.

```
    (status!readwrite!partner(u)!idle!busy!BC?ser:flist; ...
 [] (status!read!partner(u)!busy!BC?ser:flist; ...
 [] (status!read!partner(u)!unobt!BC?ser:flist; ...
```

The divert always feature offers further possible behaviour for an originating call, in basic call mode and calling state. Namely, if the dialled number has the divert always *service*, then the call should be diverted. This possibility is expressed by:

```
[calling(u) and BCmode(u)] →
        (status!read!partner(u)?s:ustate!DAl?ser:flist?d:id; ...
```

The guards are in fact identical, therefore we have to consider overlaps between the subsequent (guarded) events. Since any "status" event has to synchronise with the Network process, we have to consider whether that process can offer both status!read!partner(u)!busy!BC?ser:flist and status!read!partner(u)?s:ustate!DBa?ser:flist. Inspection of the network process reveals that it offers

```
[inDBa(mode(u),cstate(u))inDBa(mode(u),cstate(u))] →
    (status!read!id(u)!state(u)!DBa!service(u)!diversion(u);...
[]
[inBC(mode(u),cstate(u))] →
    (status!read!id(u)!state(u)!BC!service(u)!diversion(u); ...
```

Since the DAl feature has precedence over BC, in all states, there is no possible solution for u which can satisfy both guards. Consequently, there is no new non-determinism, and no interaction, at this point.

Our static analysis of the prototype model did not uncover any previously unknown interactions between the features considered. This is not surprising, as we have taken some care over the feature precedences. *However*, we did uncover several interactions which were the result of incorrect implementations of features (e.g. call barring) and the network manager. Interestingly, these errors had not been discovered during animation and property checking, thus the value of this kind of analysis was confirmed.

8 Discussion

8.1 High Level Presentation

At the higher level, we have made features, and the orderings between features as "first-class" concepts, according to the modelling principle that sources of difficulty (e.g. conflicts between features) should be made as explicit as possible. This has helped us achieve two goals: avoiding replication of behaviour descriptions when developing new features, and "capturing" certain kinds of interactions by logical inconsistencies and/or non-determinism in the model. While analysis of overlapping guards is not a new idea, as an interaction analysis technique within the context of a LOTOS model, it does appear to be a novel. Furthermore, feature theories allow us to experiment with feature precedences, and "design away" classes of interactions. Of course, over-specification of these theories may hide potential interactions.

The call for non-monotonic extensions to network behaviour is addressed by employing the LOTOS operator for choice (this operator is not monotonic with respect to the testing relation *red*, i.e. while P *red* (P []Q) – P responds to tests in the way that P [] Q responds – the converse does not necessarily hold).

We note that the authors of [8] also employ LOTOS for modelling processes and guarded choice to model some aspects of (what they refer to as policy) feature behaviour. In their case, though, the predicates simply model subscription to a feature. While interactions are not explicitly addressed in [8], it is very interesting to note that two such similar modelling approaches were developed independently.

Finally, the high level presentation may also provide the underpinning for an informal, perhaps graphical notation used in a service creation environment. Namely, it could provide a common notation linking a service creation environment and the low level model, prompting the developer to consider the appropriate states and synchronisations. The abstract properties could provide a good starting point for the natural language descriptions of the features of a service, many of which are currently incomplete and very ambiguous. Moreover, experience with high level presentations and a variety of feature theories may inform the development of algorithms for on-line resolution techniques. In particular, we may be able to encapsulate feature precedences by action sequences, which can then be detected at run-time. Such an approach may be necessary when interfacing to undocumented legacy systems.

8.2 Conclusions

The consequences of interworking inconsistencies, or feature interactions, are simply expressed, yet the sources are notoriously difficult to define and resolve. Formal models may help us to get a "handle" on some of the complex problems involved.

We have described a three level modelling approach – abstract properties, LOTOS description, and transition system. The approach allows the systematic detection and resolution of certain classes of feature interactions. We have tried to find a level of abstraction which both reflects some aspects of implementations as well as a high-level task analysis from users' viewpoints.

Like other formal approaches, we can uncover and resolve predictable interactions, as encapsulated by specific abstract, temporal properties, through analysis of the (lower level) model. In our case, this is automated through prototyping and model-checking.

Perhaps more interesting, we can also uncover further interactions through systematic, static analysis of the high level presentation of the model. These interactions can be resolved, or "designed away" through redefinition of the feature precedences. An issue for further work is to quantify the class of interactions which can be detected and resolved in this way.

While we found that a process algebra with data theories, such as LOTOS, provided the right abstractions for the high level presentation, it may be necessary, in order to employ more effective model-checking, to employ a lower level process description language. This conjecture needs to be confirmed though further experimentation.

Acknowledgements

This work has been supported by Softlab, BT Laboratories, Martlesham. I particularly thank Ben Strulo, Tim Regan and Richard Kett from Softlab for their help, valuable advice, and probing questions.

References

[1] Advanced Personal Network Services, Product Breakdown. British Telecom Laboratories. 1992.

[2] A. Alfred, N. Griffith. Feature Interactions in the Global Information Infrastructure. In Proceedings of 3rd ACM Sigsoft Symp. on Foundations of Software Engineering, *Software Engineering Notes*, Vol. 20, No. 4, Oct. 1995.

[3] P. Combes, S. Pickin. Formalisation of a User View of Network and Services for Feature Interaction Detection. In [5].

[4] Proceedings of International Workshop on Feature Interactions in Telecommunications Systems II, St. Petersburg, U.S.A., IEEE Communications Society, 1992.

[5] W.Bouma and H.Velthuijsen (eds.). *Feature Interactions in Telecommunications Systems II.* Proceedings of International Workshop, Amsterdam, IOS Press, 1994.

[6] K.E. Cheng and T. Ohta (eds.). *Feature Interactions in Telecommunications Systems III.* Tokyo, IOS Press, 1995.

[7] E.J. Cameron, N.D. Griffeth, Y.J. Lin, M.E. Nilson, W.K. Shnure, and H. Velthuijsen. A feature interaction benchmark in IN and beyond. In [5].

[8] M. Faci, L. Logrippo, and B. Stepien. Structural Modals for Specifying Telephone Systems. To appear in *Computer Networks and ISDN Systems.*

[9] H. Garavel et al. CAESAR toolkit. Available from Hubert.Garavel@imag.fr.

[10] M. Hennessy and R. Milner. Algebraic Laws for Nondeterminism and Concurrency. *Journal of the Association for Computing Machinery*, 32(1):137–161, 1985.

[11] *Information Processing Systems – Open Systems Interconnection – LOTOS – A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour.* International Organisation for Standardisation. 1988.

[12] F.J.Lin and Y-J. Lin. *A Building Block Approach to Detecting and Resolving Feature Interactions*, in [5].

[13] G. Holzmann. Design and Validation of protocols: a tutorial. *Computer Networks and ISDN Systems*, No. 25, pp. 981-1017, 1993.

[14] J. Quemada, S. Pàvon, and A. Fernandez. Transforming LOTOS specification with LOLA - The Parameterised Expansion. In K. Turner, editor, *Formal Description Techniques, I.* Elsevier Science Publishers B.V. (North-Holland), 1988.

[15] [LOLA 94] J.Quemada, S. Pàvon, D. Larrabeiti. LOLA User Manual. Dpt. Ingenieria Telematica, E.T.S.I. de Telecommunicacion, Univ. Politecnica de Madrid, 1994.

[16] C. Stirling. An Introduction to Modal and Temporal Logics for CCS. In F. Moller, editor, *Logics for Concurrency: Structure vs Automata*, 1994. The VIIIth Banff Higher-Order Workshop. Banff, Alberta, Canada.

[17] H.Velthuijsen. Issues of Non-monotonicity in Feature-Interaction Detection. In [6].