Avoiding Feature Interactions in the Users' Interface

Jan Bredereke

McMaster University, CRL
1280 Main Street West · Hamilton, Ontario L8S 4K1 · Canada
jan.bredereke@poboxes.com · http://www.poboxes.com/jan.bredereke

Abstract.
The causes for feature interactions are many, therefore several remedies will have to be applied together. In this paper, we investigate the architecture of the users' interface of a telephone switching system and tackle a certain kind of feature interactions by employing a more modular structure for the design of the system and its features. We propose to apply results from the research on human-computer interfaces (HCI) to the specification of requirements of telecommunication systems. After we have confined the feature interactions in question to one information-hiding module, we show what is necessary to detect them by formal validation. We discuss briefly how the general approach can be applied to the existing Intelligent Network (IN) architecture in particular.

1 Introduction

Feature interactions in telecommunication systems increasingly obstruct the introduction of new features. They are an enstablished research subject by now (compare, e.g., [1, 2, 3]). The causes for feature interactions are many, therefore many remedies will have to be applied together. We aim at tackling a certain kind of feature interaction by employing a more modular structure for the design of a telephone switching system and its features. In this paper, we the architecture of investigate the users' interface of such a system.

We propose to separate the concern of the users' interface more clearly from the concern of the behaviour of features, in particular when new features are introduced into a telephone switching system. Of course, this helps to handle complexity by a divide-and-conquer technique, but in particular it enables us to validate formally the absence of a certain kind of feature interaction in a small part of the entire system, while we avoid this kind of feature interaction entirely in the remaining parts.

As motivation, we present one example for a feature interaction that is caused by the way the users' interface often is expressed. It can occur between a *Credit Card Calling* (CCC) feature and a *Voice Mail* feature. The example is taken from the feature interaction benchmark by Cameron et. al. [4]. Credit Card Calling includes an authorization phase where the card's number and PIN must be entered. For convenience, this feature often allows placing another call without re-entering, by just pressing the "#" button at the end of the first call. On the other hand, to access voice mail messages from phones other than his own, a subscriber of some voice mail service can call his own phone number (maybe using the CCC feature) and then press the "#" button followed by some identification to indicate that he is the subscriber. These features can't work together properly since the telephone system has no way to determine whether pressing the "#" button means that the caller just doesn't want to leave a message, or whether he is the subscriber.[1]

The problem in the example is that two features are active concurrently, but they are defined in a way that assumes that each of them is the only feature communicating via the associated user interface. There is no general provision that controls the access to this resource by different features. Today, hundreds of features have been conceived, with many of them requiring user interaction. Nevertheless, many features could be used concurrently and independently, if the users' interface resource, and in particular, the signals that a user can generate (or receive) on a standard phone, were not so limited.

There are some more examples of feature interactions in the benchmark [4] that are caused in this way: *Call Waiting* and *Three-Way Calling* (ambiguous flash hook), *Multi-location Business Service-Extension Dialing* and *CENTREX* (ambiguous numbering plan), *Call Waiting (CW)* and *Personal Communication Services (PCS)* (definition of CW for PCS user X ignores that a phone may be shared with PCS user Y not subscribing to CW), *Call Waiting* and *Call Waiting* (a user may take part in two concurrent CW contexts; hanging up can be ambiguous with respect to ringing back for held parties, if both contexts are in different modes), and *Call Waiting* and *Three-Way Calling* (ambiguous hang up in a complex context).

Still, features are often described as if each of them had exclusive control of the users' interface. Notably, the Feature Interaction Detection Contest instructions [5] take exactly this approach. They specify features in terms of "Flash", "Off-hook", "DialTone", "StartRinging" etc. Many of the case study specifications in the proceedings of previous feature interaction workshops [1, 2, 3] are expressed similarly. Even though they aggregate individual digits to a "dialled number", in general these requirement specifications of features make assumptions about details of the users' interface hardware.

There are some exceptions in the literature, for example Blom et al. [6] assume a "parser" that receives signals from the outside (e.g., flash hook), and delivers abstract events, such as "Transfer", "Switch", and "3Way", depending on the state of the system. Other exceptions are Gibson [7] who recommends to distinguish clearly between abstract actions/signals in the requirements model and concrete actions/signals in the implementation model, and Zibman et. al. [8] who propose to use "logical names" instead of "specific signals". Even though Cattrall et. al. [9] do not address low-level signals, they work with abstract signals entirely. Griffeth and Velthuijsen [10] attempt to derive the intentions behind low-level signals by a goal profile stored in an automated negotiating agent.

The Intelligent Network (IN) standard [11, 12] is intended to facilitate the introduction of new features. Nevertheless, it prescribes only partially how user interactions with these features are processed, and thus there is no general, coordinated way for user interaction processing beyond the provisions for basic call processing. Partially, this is alleviated by imposing restrictions on a concurrent activation of features, but this prevents many useful features from being specified. These restrictions have been reduced in the IN capability set 2 (CS-2) [13], but in the IN CS-2 architecture there is still no mandatory synchronization protocol among the features concerning user interaction.

In Section 2, we sketch how most of the requirements and the design of a feature can

[1]This feature interaction is resolved in the calling card feature to which the author subscribes: the "#" button must be pressed at least 2 seconds to take effect. Thus, we have two distinguishable signals.

be specified precisely, without refering to any hardware details, and how these details can be encapsulated. In Section 3, we apply this to the Intelligent Network architecture. In Section 4, we show that the approach allows validation of the encapsulated users' interface for the absence of a certain kind of feature interaction. Section 5 summarizes the results and gives an outlook.

2 Encapsulating the Users' Interface

2.1 Specifying Feature Requirements on the Semantic Level

A human-computer interface allows a human and a machine to interact; for this, they must use a language. As in other areas concerned with languages, one can distinguish the syntax and the semantics. The model of Foley and Wallace [14, 15] (compare also [16]) is often used for (graphical) human-computer interfaces. They distinguish two major components: the form and the meaning. For their kinds of interfaces, they continue and differentiate more levels. For the purposes of this paper, we only need the distinction between the specific form of an interaction, i.e., the syntax, and its meaning, i.e., the semantics.

The lessons learned with the human-computer interface can and should be applied to the user-telephone interface, too. We propose to specify the requirements for a telephone switching system, including any additional features, on the semantic level. For the moment, we assume that we are interested in the requirements of an entire such system, and not only a part of it. If we specify the behaviour of the system solely on the semantic level, then all the feature interaction problems listed in Section 1 simply disappear from the requirements document, since they have become a design problem.

In the example of Credit Card Calling (CCC) and Voice Mail (VM), we now have separate sets of semantic signals from the user to the system for each of the features. The CCC feature adds something like a *ReleaseAndReconnect* semantic signal to the system, and the VM feature adds something like a *VMlogin* semantic signal. If the functionality of two features is not related, they should always have separate sets of semantic signals.

We propose that a semantic signal should reflect (a) a *user's decision* to perform some action, for example to request a communication connection to some other user, (b) or a *user's perception* that some other user (or the system) has decided to perform some action, for example to request a communication connection with him.

But this move has not yet solved the problem. Eventually, all of the semantic signals must still be mapped onto the few syntactic signals, and back. The exact mapping is a concern of the users' interface, and it has little to do with the functionality of the telephone switching system and its features. According to the well known information-hiding principle (e.g., [17]), it should therefore be encapsulated as a secret of only one design module, which we call the *users' interface module* (UIF). It will still be difficult to find a satisfactory mapping. But if all aspects of this problem are separated and concentrated into a single module, it will be easier to understand its semantics, and thus to conceive a solution, and the module will be easier to change later. The latter is of particular importance to the feature interaction problem. Furthermore, it will be easier to validate the correctness of a solution, as we will see in Section 4.

We have to stress that an information-hiding module such as the UIF module is *not* necessarily implemented by some centralized software program or in a single hardware component [17]. The different access functions, for example the mapping routines, may be distributed over many software packages and to many physical locations. But the module must be *maintained* as a single unit, with any changes distributed in a consistent manner. We will discuss this issue further in Section 3.

Currently, we are performing a case study [18] in which we are specifying a telephone network on the semantic level, or at least some of its functionality, using the Functional Documentation approach [19, 20, 21, see Section 2.3].

2.2 Formal Distinction of the Pre-Existing Environment

Usually, we are not in a position to design the requirements for an entirely new telephone switching system. For example, we might not have a choice of the hardware that the users will use. Similarly, we might be required to use pre-existing software, in particular some telephone switching system with a software interface conforming to the Intelligent Network recommendations [12, 13]. Nevertheless, we *can* express the requirements on any part of the system, in particular, on new telephony features, on the semantic level.

The requirements document must contain additionally a description of the relevant aspects of the pre-existing parts. The implementer should realize the semantic-level specification by making use of these pre-existing means. In the requirements document, the assumed environment should be distinguished clearly from the new system to construct, such that the document is a self-contained, complete description. We are interested, in particular, in formal requirements documents. They avoid ambiguities and allow for validation (compare Section 4). Therefore, the distinction between the system to construct and the environment to assume should be formal, too.

Please note that the boundary between the system to construct and its environment usually is *not* the users' interface; these are two completely different kinds of boundaries, which should not be confused.

2.3 Formal Specification Technique "Functional Documentation"

In addition to identifying all environmental entities that are of relevance, and distinguishing them, a formal requirements specification must indicate whether the system to construct can control these environmental entities or not. For example, the user's decision to release and reconnect cannot be controlled, and neither can the triggering of Detection Points (DPs) by the underlying basic call processing software. But the system can control the arming of a DP, and it can control procedure calls to the underlying software.

Any entities that are only to be constructed are *not* part of the formal requirements specification. They are part of the design. For example, the requirements specification must not prescribe any local state components, e.g., Personal Identification Number (PIN) lists, if these entities do not exist before the system is constructed. The required behaviour of a system to construct should be stated only in terms of the history of its environment's behaviour.

Functional Documentation [19, 20, 21] is a specification technique that allows formal specification of the requirements of a system in terms of environmental entities only. Furthermore it distinguishes them clearly into *controlled variables* and *monitored variables*, denoted by two vectors of time-functions $\underset{\sim}{c}^t$ and $\underset{\sim}{m}^t$, respectively. The environment to be assumed is specified as a relation *NAT*. NAT describes how the relevant part of the world may behave over time without or despite the system to construct: $\mathsf{domain}(\mathrm{NAT})$ exactly contains the instances of $\underset{\sim}{m}^t$ allowed by the environmental constraints, and $\mathsf{range}(\mathrm{NAT})$ exactly contains the instances of $\underset{\sim}{c}^t$ allowed by the environmental constraints. $(\underset{\sim}{m}^t, \underset{\sim}{c}^t) \in \mathrm{NAT}$ if and only if the environmental constraints allow the controlled quantities to take on the values described by $\underset{\sim}{c}^t$, if the values of the monitored quantities are described by $\underset{\sim}{m}^t$. The world, as it may behave with the system present, is described by the intersection $\mathrm{NAT} \cap \mathrm{REQ}$. The relation REQ may restrict the behaviour of controlled variables only, but not of monitored variables: $\mathsf{domain}(\mathrm{REQ})$ exactly contains the instances of $\underset{\sim}{m}^t$ allowed by the environmental constraints, and $\mathsf{range}(\mathrm{REQ})$ exactly contains the instances of $\underset{\sim}{c}^t$ allowed by a correct system. $(\underset{\sim}{m}^t, \underset{\sim}{c}^t) \in \mathrm{REQ}$ if and only if the system should permit the controlled quantities to take on the values described by $\underset{\sim}{c}^t$ when the values of the monitored quantities are described by $\underset{\sim}{m}^t$.

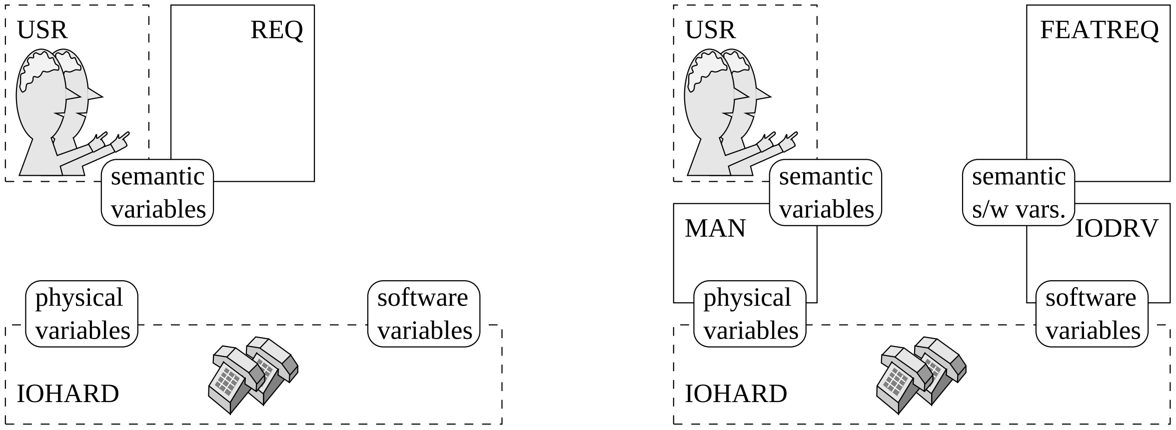
In this paper, we will also employ relations over subvectors of $\underset{\sim}{m}^t$ and $\underset{\sim}{c}^t$, and over vectors of variables $\underset{\sim}{v}^t$ introduced during design. Thus, relations become ternary, with REQ and NAT having an empty vector of design variables $\underset{\sim}{v}^t$ as a special case. For a relation R, we generalize $\mathsf{domain}(R)$ to $\mathsf{mon}(R)$, $\mathsf{range}(R)$ to $\mathsf{ctrl}(R)$, and we add $\mathsf{des}(R)$ to designate the third design variable vector. We assume some total ordering over each kind of variables, and further we assume that all vectors of variables have their components sorted. We denote the restriction of a triple of variable vectors $r = (\underset{\sim}{m}^t, \underset{\sim}{c}^t, \underset{\sim}{v}^t)$ to the variables of a relation R as $r{\downarrow_{\mathrm{R}}}$ (R must comprise no other variables than those of the vector triple r). For two relations R, S, we define the relation $\Theta_{R,S}$ as the relation that contains exactly those variables in $\mathsf{mon}(\Theta_{R,S})$, $\mathsf{ctrl}(\Theta_{R,S})$, and $\mathsf{des}(\Theta_{R,S})$ that appear in at least one of R or S in the respective vectors, with $\Theta_{R,S}$ being the largest relation doing so. This allows us to define the composition of relations $R \hat{\cap} S \stackrel{\mathrm{def}}{=} \{r \mid r \in \Theta_{R,S} \wedge r{\downarrow_{\mathrm{R}}} \in R \wedge r{\downarrow_{\mathrm{S}}} \in S\}$.

2.4 Explicit User Manual Relation MAN in Design

Now, we discuss how the semantic signals can be related to the pre-existing entities of the users' interface. For the moment, we assume that we design a single feature only, with no other features already in the environment. In Section 2.5 below, we remove this restriction and discuss the general case of many features.

Figure 1(a) shows the components of the requirements document. We group the variables (both monitored and controlled) into three vector triples, as shown. The first, the vector triple of *semantic variables*, describes the semantic signals by the users. Furthermore, we split the description of the relevant environment into two relations. The first relation *USR* describes what the users might do on the semantic variables (USR is strongly non-deterministic since many aspects of user behaviour are not relevant for the system). The second relation *IOHARD* describes some pre-existing input/output hardware (and maybe software), by relating a vector triple of *physical variables* to a vector triple of *software variables*. The requirements of the feature are described by the relation REQ. IOHARD and the low-level physical and software variables have not yet any connection to REQ. We deviate from the standard approach insofar as we define REQ only over the semantic variables, not over all variables mentioned. Nevertheless, $(\mathrm{USR} \hat{\cap} \mathrm{IOHARD}) \hat{\cap} \mathrm{REQ}$ still describes the allowed behaviour of the system, using the definition of composition from the previous section. The entire environment NAT can be derived from the composition of USR and IOHARD:

$$\mathrm{NAT} \stackrel{\mathrm{def}}{=} \mathrm{USR} \hat{\cap} \mathrm{IOHARD}$$



(a) Requirements document.

(b) Design document.

legend: environment relation system relation variables

Figure 1: Structure of the documents.

In Figure 1(b), we propose a *design structure* for these requirements that encapsulates the users' interface. There are two important aspects in this design. First, we introduce a software reflection of the semantic variables (which cannot be observed directly by any machine). These design variables, the *semantic software variables*, separate the software into an input/output driver part *IODRV* and another version of the feature's requirements *FEATREQ*. FEATREQ is usually very similar or identical to REQ, modulo variable renaming; the difference is that the semantic software variables can be observed by the software, while the semantic variables in the users' minds cannot.

The second important aspect is how the semantic variables in the users' minds are related to the physical variables in the design. We propose to describe this relation by an explicit relation *MAN*. It documents explicitly how the users translate their decisions on the semantic level into physical actions, and how physical actions translate to users' perceptions. These translations do not exist before the system is designed, therefore MAN is part of the design. The system designer conceives these translation rules, and usually he documents them in a user manual (hence the name MAN).

For the validation purposes in Section 4, we need to write down this relation explicitly and formally. It should be noted that the formal document supports validation, but we should also provide an informal version that is structured differently, which the users can understand easily.

The users' interface relation, for short *UIF*, is composed of the relations MAN, IOHARD, and IODRV:

$$\mathrm{UIF} \stackrel{\mathrm{def}}{=} \mathrm{MAN} \hat{\cap} \mathrm{IOHARD} \hat{\cap} \mathrm{IODRV}$$

The users' interface relation is a part of the design, since some of its components do not exist before the system is constructed. The users' interface UIF is a *module in the information-hiding sense*, as discussed in Section 2.1. Its *secret* is how the high-level semantic variables are mapped to the low-level syntactic physical and software variables. This encapsulation becomes possible by introducing an explicit user manual relation MAN.

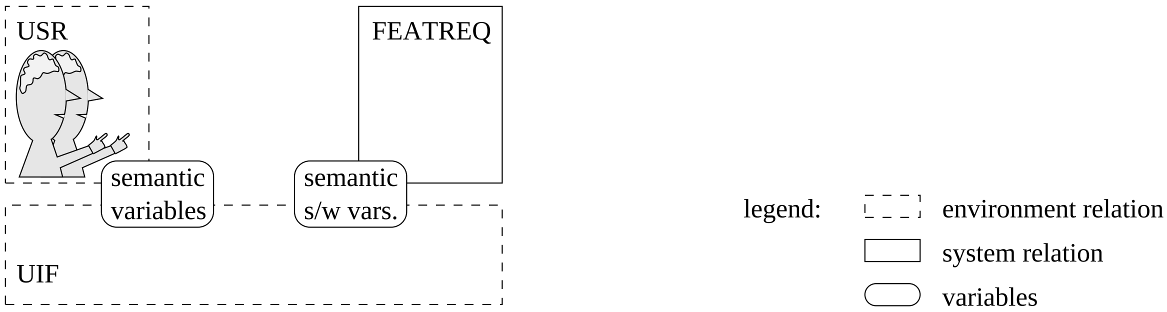


Figure 2: Design with a users' interface that maps semantic variables one-to-one onto semantic software variables.

Overall, all the relations in Figure 1(b) with solid lines together form the design $DSGN$ and must implement the relation REQ in Figure 1(a), in the given environment. Formally, we write this as:

$$
\begin{array}{lll}
\multicolumn{3}{l}{(\mathrm{DSGN} \hat{\cap} \mathrm{NAT}) \mathrel{\hat{\subseteq}} (\mathrm{REQ} \hat{\cap} \mathrm{NAT})} \\
\text{with} & \mathrm{NAT} \stackrel{\text{def}}{=} & \mathrm{USR} \hat{\cap} \mathrm{IOHARD} \\
\text{and} & \mathrm{DSGN} \stackrel{\text{def}}{=} & \mathrm{MAN} \hat{\cap} \mathrm{IODRV} \hat{\cap} \mathrm{FEATREQ}
\end{array}
$$

Analogously to the composition $\hat{\cap}$, we define the variable-extended implication predicate $R \hat{\subseteq} S$ (for relations S that have no design variables)

$$
\begin{array}{ll}
R \hat{\subseteq} S \stackrel{\text{def}}{=} & (\forall r \in R : r{\downarrow_{\mathrm{S}}} \in S) \wedge \\
& (\Theta_{R,S} = \Theta_{R,R}) \wedge (\{\underset{\sim}{m}^t{\downarrow_{\mathrm{S}}} \mid \underset{\sim}{m}^t \in \mathsf{mon}(R)\} = \mathsf{mon}(S))
\end{array}
$$

2.5 Coordinating the Encoding of Semantic Signals

We have seen how the users' interface aspects of *one* feature can be encapsulated in the design. But how can we coordinate this for many features, and with the base system? The answer is, of course, that there must be one single users' interface module in the environment of all these features, and the concerns of the users' interface must not be part of the design of the functionality of the features at all. That means that we have to provide once and for all an information-hiding module that provides a one-to-one relation between the semantic signals in the mind of the users and a representation of them as semantic software signals, as shown in Figure 2.

The users' interface module must be *extensible*; the module must allow pairs of semantic mental variables and semantic software variables to be added to its interface. Now, we have split the job of implementing the requirements of a new feature into two parts: implementing the behaviour of the feature, and extending the users' interface module, if necessary.

We have to stress that, as already mentioned in Section 2.1, a module in the information-hiding sense does not need to be implemented in one centralized spot.

Adding new semantic variables to the users' interface module may be done either offline during the feature design process, or it may be done dynamically by an online variable registration facility in the users' interface. When a designer extends the user interface module offline, he can consider "good" human-computer interface design, and he can perform extensive validation of the correctness of his result, as discussed in Section 4 below. The drawback of offline extension is that all features that will access the users' interface must be known beforehand. In a dynamic and heterogenous network, this might prove difficult.

Alternatively, we can design a set of generic semantic variables beforehand, for which we design a workable mapping to the syntactic variables, and then let an automated agent within the users' interface module assign these generic variables to arbitrary features on request. This assignment process must include a suitable online instruction of the affected users, for example announcements. If we decide for an automated registration agent, the users' interface module must comprise some special variables that features can use to issue registration requests.

There has already been some work in the above direction. Zibman et. al. [8] take a more general view and propose an architecture out of agents that separates concerns, in conjunction with a processing model. The architecture includes a terminal agent, which, amongst others, manages the dynamic assignment of signals to a new service. Since the authors focus more on general aspects, this is not detailed further; also, they present no formal model of their agent architecture. Keck [22] proposes a service admission control scheme for the IN that keeps track of the resources relevant to a session and that notifies services in case of possible interactions. Also related is the work on Open Distributed Processing (ODP) [23], which attempts to identify several useful abstractions and to separate several communication-related concerns. Van der Linden [24], for example, proposes to use an ODP architecture to beat feature interactions, and the TINA initiative [25] customizes the ODP ideas for telecommunication networks.

Our discussion on extending the users' interface module did not comprise the question of how the functionality of the telecommunication system should be extended, i.e., how the relation FEATREQ in Figure 2 relates to and modifies the base system and other features. This is outside the scope of this paper, it is basically the question of how to tackle feature interactions not related to the users' interface.

3 Application to the Existing IN Architecture

How can we apply the above general principle to a concrete framework like the IN architecture?

The IN standard allows to build new services on top of the basic call processing by using Service Independent Building Blocks (SIBs), and by connecting these with so-called Service Logic [12]. One of the SIBs defined in the IN CS-1 standard is a "user interaction" SIB. The standard states that this SIB provides a call party with information (e.g. announcements) and/or collects information from a call party. The announcements can be, for example: DTMF tones, a customized or generic audio message, or network progression tones (e.g. dial tone, busy tone, etc.). The collected information can be, for example: DTMF tones, audio, or IA5 String text [26]. The parameters of this SIB include, among others, the announcement ID, the number of repetitions, and the number of the expected digits. Thus, the user interaction SIB clearly provides a user interface at the syntactic level, according the model of Foley and Wallace.

Furthermore, there are other SIBs that share user interface related concerns, for example "queue" (plays announcements), "verify" (performs a syntactic check of user input), and "translate" (relies, e.g., on the structure of directory numbers).

The IN SIBs are intended to be reusable software components. A suitable SIB can be used as part of an information-hiding module that encapsulates the users' interface and that provides variables at the semantic level. Such a SIB can be used as part of the environment of a new service to conceive.

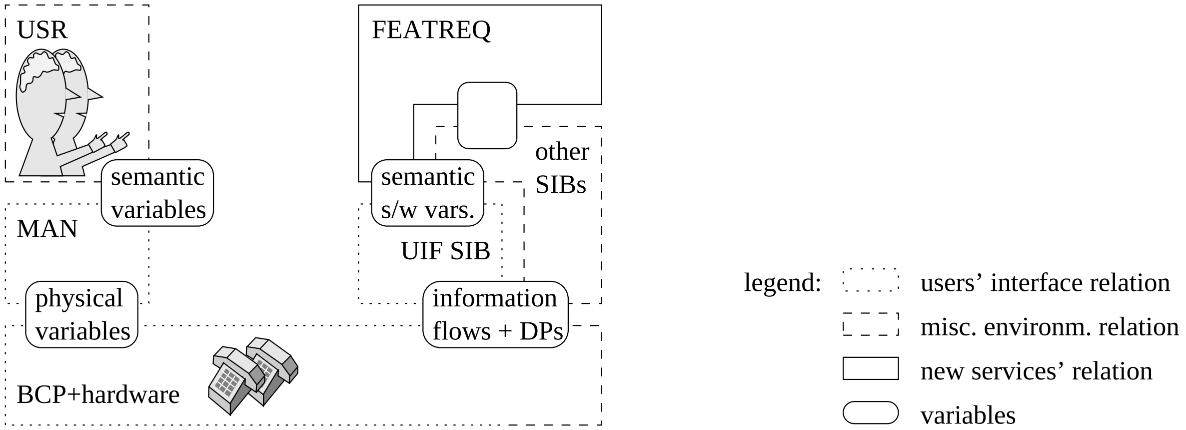


Figure 3: Encapsulated users' interface in the IN architecture.

For this, we need a redesigned set of SIBs where only the "users' interface" (UIF) SIB is concerned with the details of the users' interface. In order that other concerns can be encapsulated, too, SIBs should be designed to be built hierarchically one on another. Fortunately, the concept of hierarchical SIBs is introduced in IN CS-2 [13].

Furthermore, a UIF SIB will be so complex that its design should be broken down futher into sub-SIBs. For example, there should be a registration agent, as discussed in the previous section.

Our overall design architecture looks like Figure 3, with the "software variables" now being the Informations Flows of the Distributed Functional Plane, plus the Detection Points, which together underly the definition of the SIBs. The "IODRV" module is now the rewritten UIF SIB. The users' interface module is completed by the relevant aspects of the underlying Basic Call Process (BCP), and, of course, by the user manual relation MAN.

A certain difficulty with the concept of the Intelligent Network arises from its separation of the Basic Call Process and of IN service processing. The Basic Call Process performs user interaction, too, which can be controlled by the IN services only partially. Only when this user interaction leads to triggering a detection point, for example when a dialled number during call setup is completed, the IN service processing can notice and influence this user interaction. Furthermore, the behaviour of the Basic Call Process between the detection points cannot be changed by the IN service processing, which makes changes to the syntax of the users' interface hard. For example, when a user goes on-hook, this can induce some irreversible state changes before the next detection point is triggered, thus we cannot reassign an arbitrary meaning to this syntactic signal. Therefore, the translation between syntactic and semantic signals has less degrees of freedom, leaving even less freely usable syntactic signals for new services.

Nevertheless, we can design a set of generic semantic variables for the users' interface module with a workable mapping to the syntactic variables, as proposed above. But any service requesting dynamically for the assignment of such a generic variable must be prepared that no more variables are available, and the service must react by a graceful degradation. Alternatively, we might provide an arbitrary number of generic semantic variables, but with increasingly reduced quality of usage comfort, such that a service has to decide whether to accept an assignment offered by the registration agent, or to degrade gracefully, again.

The semantic variables of the users' interface are each associated to only one user, since they reflect mental processes. Therefore, the corresponding syntactic physical variables of one user are associated to one physical Service Control Point (SCP) only. As long as a user always employs the same line, all users' interface processing related to him can be performed at the same location. As soon as a user is allowed to roam and to use his IN services on different lines, the UIF SIB becomes distributed. Now, the implementation of the UIF SIB incurs substantial additional organizatorial and technical complexity. Nevertheless, the UIF SIB still should be regarded as a single module, and its problems should be separated from other concerns of IN service functionality. In this case, the UIF SIB certainly has a sub-structure. For example, the variable registration agent can be a sub-module of its own, encapsulating the assignment strategy for the generic semantic variables. We have not yet specified an example UIF SIB in our ongoing case study [18], this is for future work.

4 Validation of the Users' Interface

Modelling a feature's behaviour at the semantic level, and separating the concerns of the users' interface, additionally allows for a formal validation specifically of the users' interface. The design specification for the feature's users' interface states a set of one-to-one correspondences between semantic signals in the mind of the users and a representation of them as software signals (compare Figures 1(b), 2, and 3). A successful design of the extended users' interface must provide a pair of mutually inverse mapping relations to syntactic signals. This task may be difficult when there are few syntactic signals. But the design can be verified formally, and iterated until it is proven correct.

There are two kinds of properties that we can verify. The first is whether the design indeed implements the *functional requirements* we have stated formally, that is, whether there is indeed a one-to-one mapping of mental to software semantic signals. Possible errors include (a) a *translation to an incorrect*, i.e., another, semantic *signal*, when the two mapping functions MAN and IODRV do not fit properly, and, in particular, (b) a *non-deterministic interpretation* of syntactic signals, which constitutes one of the dreaded signalling ambiguities.

Even without having an explicit relation MAN, we can do a check for a non-deterministic interpretation, by generating and checking all permutations of syntactic signals. This is a common approach [1, 2, 3], but a disadvantage is that it may lead to false warnings, since there are (possibly many) state/signal combinations that are reachable only when the system is not used according to its usage rules, and the users then should not expect the system's features to work as intended anyway.

The second kind of properties of the users' interface design that we can validate are on *usability*. In the area of human-computer interface research, there have been identified numerous properties which are desirable or undesirable (cf., e.g., [16]), and several of them can be checked even formally [27]. For example, (a) *almost-alike states* of the users' interface [28] tend to confuse users, (b) *interactive deadlock* [29] prevents users from leaving a state due to (in particular) insufficient help available, and (c) the amount of *character level ambiguity* [30] should be minimized, which is a measure derived from the number of different modes in which each keystroke has different meanings. The latter optimization is of particular interest in our context, where there are many more semantic signals than syntactic signals.

In this paper, we are only concerned about how to gather the information necessary for a validation step, as one of its preconditions. We do not discuss how to perform an actual validation step; this is a wide area of research itself [1, 2, 3].

5 Summary and Outlook

The causes for feature interactions are many, therefore many remedies will have to be applied together. In previous work we investigated the detection and resolution of feature interactions by formal methods and tool support [31, 32]. This rendered promising results; nevertheless, one should try to avoid problems in the first place, if one can. Therefore, in this paper we aim at tackling a certain kind of feature interaction by employing a more modular structure for the design of a telephone switching system, and its features.

We proposed to apply results from the research on human-computer interfaces (HCI) to the specification of requirements of telecommunication systems. In particular, these requirements should not be expressed on the level of syntactic variables, such as "flash hook" or certain button presses, but in terms of semantic variables such as "Release-AndReconnect" (for Credit Card Calling). A semantic variable reflects a decision of a user to perform some action, or a perception of a user of someone else's action. We are currently performing a case study that follows this approach [18], using the Functional Documentation method [19, 20, 21] to specify the requirements.

In this way, feature interaction problems related to the users' interface disappear from the requirements document and become a design problem. For the design, we proposed to separate concerns and proposed to introduce an information-hiding module encapsulating the details of the translation of semantic signals to syntactic signals. This confines the feature interaction problems related to the users' interface to this module only. In contrast, for example the Feature Interaction Detection Contest instructions [5] mingle these problems freely with all kinds of other problems, thus increasing complexity and making solutions harder.

We also briefly discussed how the general approach can be applied to the existing Intelligent Network (IN) architecture in particular. Finally, the encapsulation of the details of the users' interface, together with an entirely formal description of it, provides the necessary description that enables a formal validation of the users' interface, in particular for feature interaction problems related to it.

In recent work, Miller et. al. have specified the mode logic of a flight guidance system in CoRE and SCR (which are variants of Functional Documentation), and they have found that the requirements specification of the user interface should have been separated in quite a similar way as proposed here [33]. More results will become available later this year.

The users' interface is only one of the resources of a telecommunication system. Therefore, a telecommunication system architecture that avoids (at least many) feature interaction problems should encapsulate the details of other resources, too. More work on this is necessary. The advent of a host of new features of telecommunications systems, which we expect to work concurrently and independently, is with some respects comparable to the concurrency and its problems when multi-user time-sharing processing was introduced for computer operating systems. The basic ideas of encapsulation developed at that time are not new anymore, but important tasks are now (a) to apply this knowledge to the specific structure of telecommunication systems, while they change and become more and more complex; and (b) to combine this with formal methods and the tool support these methods provide.

References

[1] Petre Dini, Raouf Boutaba, and Luigi Logrippo, editors. "Feature Interactions in Telecommunication Networks IV". IOS Press, Amsterdam (June 1997).

[2] Kong Eng Cheng and Tadashi Ohta, editors. "Feature Interactions in Telecommunications III". IOS Press, Amsterdam (1995).

[3] L. G. Bouma and Hugo Velthuijsen, editors. "Feature Interactions in Telecommunications Systems". IOS Press, Amsterdam (1994).

[4] E. Jane Cameron, Nancy D. Griffeth, Yow-Jian Lin, Margaret E. Nilson, William K. Schnure, and Hugo Velthuijsen. A feature interaction benchmark in IN and beyond. In Bouma and Velthuijsen [3], pages 1–23.

[5] Nancy Griffeth, Ralph Blumenthal, Jean-Charles Gregoire, and Tadashi Ohta. "Feature Interaction Detection Contest, 5th Int'l. Workshop on Feature Interactions – Instructions" (1998). URL: http://www.tts.lth.se/FIW98/contest.html.

[6] Johan Blom, Roland Bol, and Lars Kempe. Automatic detection of feature interactions in temporal logic. In Cheng and Ohta [2], pages 1–19.

[7] J. Paul Gibson. Feature requirements models: Understanding interactions. In Dini et al. [1], pages 46–60.

[8] Israel Zibman, Carl Woolf, Peter O'Reilly, Larry Strickland, David Willis, and John Visser. Minimizing feature interactions: an architecture and processing model approach. In Cheng and Ohta [2], pages 65–83.

[9] D. Cattrall, G. Howard, D. Jordan, and S. Buf. An interaction-avoiding call processing model. In Cheng and Ohta [2], pages 85–96.

[10] Nancy D. Griffeth and Hugo Velthuijsen. The negotiating agents approach to runtime feature interaction resolution. In Bouma and Velthuijsen [3], pages 217–235.

[11] José M. Duran and John Visser. International standards for Intelligent Networks. *IEEE Commun. Mag.* **30**(2), 34–42 (February 1992).

[12] ITU-T. "Q.12xx-Series Intelligent Network Recommendations" (1993).

[13] ITU-T. "Q.122x-Series Intelligent Network Recommendations for Capability Set 2" (1997).

[14] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. "Computer Graphics: Principles and Practice". Addison-Wesley, 2nd edition (1990).

[15] James D. Foley and Victor L. Wallace. The art of natural graphic man-machine conversation. *Proc. of the IEEE* **62**(4), 462–471 (April 1974).

[16] Ben Shneiderman. "Designing the User Interface: Strategies for Effective Human-Computer Interaction". Addison-Wesley, 3rd edition (1997).

[17] David Lorge Parnas. On the criteria to be used in decomposing systems into modules. *Commun. ACM* **15**(12), 1053–1058 (1972).

[18] Jan Bredereke. Requirements specification and design of a simplified telephone network by functional documentation. CRL Report NNN, McMaster University, Hamilton, Ontario, Canada (1998). *To appear.*

[19] David Lorge Parnas and Jan Madey. Functional documents for computer systems. *Sci. Comput. Programming* **25**(1), 41–61 (October 1995).

[20] A. John van Schouwen, David Lorge Parnas, and Jan Madey. Documentation of requirements for computer systems. In "IEEE Int'l. Symposium on Requirements Engineering – RE'93", pages 198–207, San Diego, Calif., USA (4–6 January 1993). IEEE Comp. Soc. Press.

[21] A. John van Schouwen. The A-7 requirements model: Re-examination for real-time systems and an application to monitoring systems. CRL Report 242, McMaster University, TRIO, Hamilton, Canada (February 1992).

[22] Dirk O. Keck. Requirements and a proposal for the prevention of a class of service interactions in Intelligent Networks. In Tiziana Margaria et al., editors, "Services And Visualization : Towards User-Friendly Design ; ACoS'98", number 1385 in LNCS, pages 90–105. Springer (1998).

[23] Kazi Farooqui, Luigi Logrippo, and Jan de Meer. The ISO reference model for Open Distributed Processing: an introduction. *Comp. Networks and ISDN Syst.* **27**(8), 1215–1229 (July 1995).

[24] Rob van der Linden. Using an architecture to help beat feature interaction. In Bouma and Velthuijsen [3], pages 24–35.

[25] C. Abarca et al.. Service architecture. Deliverable, TINA-Consortium, URL: http://www.tinac.com/ (16 June 1997). Version 5.0.

[26] ITU-T, Recommendation Q.1213. "Global Functional Plane for Intelligent Network CS-1" (March 1993).

[27] Robert J. K. Jacob. Executable specifications for a human-computer interface. In Ann Janda, editor, "Proc. ACM CHI'83 Human Factors in Computing Systems Conf.", pages 28–34, Evanston, USA (12–15 December 1983). ACM, North-Holland.

[28] David Lorge Parnas. On the use of transition diagrams in the design of a user interface for an interactive computer system. In "Proc. 24th National ACM Conf.", pages 379–385 (1969).

[29] Jared Darlington, Wolfgang Dzida, and Siegfried Herda. The role of exursions in interactive systems. *Int'l. Jour. of Man-Machine Studies* **18**(2), 101–112 (February 1983).

[30] Harold Thimbleby. Character level ambiguity: consequences for user interface design. *Int'l. Jour. of Man-Machine Studies* **16**(2), 211–225 (February 1982).

[31] Jan Bredereke. "Communication Systems Design With Estelle – On Style, Efficiency, and Analysis". PhD thesis, Univ. of Kaiserslautern, Shaker Verlag, Aachen, Germany (August 1997). ISBN 3-8265-2764-X.

[32] Jan Bredereke. Detection of feature interactions in Intelligent Networks by verification. *Software Concepts & Tools* **17**(3), 121–139 (1996).

[33] Steven P. Miller. Specifying the mode logic of a flight guidance system in CoRE and SCR. In "Second Workshop on Formal Methods in Software Practice", Clearwater Beach, Florida, USA (4–5 March 1998).