Feature Interaction: A Critical Review and Considered Forecast

Muffy Calder [a] Mario Kolberg [b,*] Evan H. Magill [b]
Stephan Reiff-Marganiec [a]

[a] *Department of Computing Science, University of Glasgow, Glasgow G12 8QQ, United Kingdom*

[b] *Department of Computing Science and Mathematics, University of Stirling, Stirling FK9 4LA, United Kingdom*

Abstract

The state of the art of the field of feature interactions in telecommunications services is reviewed, concentrating on three major research trends: software engineering approaches, formal methods, and on line techniques. Then, the impact of the new, emerging architectures on the feature interaction problem is considered. A forecast is made about how research in feature interactions needs to readjust to address the new challenges posed by the emerging architectures.

Key words: Feature Interaction, Telecommunications Services, Incompatibility

1 Introduction

We present an overview of research work in the field of feature interactions in telecommunications services. Our aim is to provide a comprehensive introduction to the field, as well as to give a considered view of the contributions so far and our own views about directions for future research. We have not tried to present a complete picture of all the research activities in the field, for a comprehensive survey up to 1998, see [69]. Rather, we try to bring together some of the major contributions and trends, assess their impact on the problem, and point the way ahead for future research.

Much of the research in the area has been stimulated by the international series of "Feature Interaction Workshops", starting in 1992 and held at regular intervals since

* Corresponding author.
Email addresses: muffy@dcs.gla.ac.uk (Muffy Calder), mko@cs.stir.ac.uk (Mario Kolberg), ehm@cs.stir.ac.uk (Evan H. Magill), sreiff@dcs.gla.ac.uk (Stephan Reiff-Marganiec).

then. The proceedings provide a good (though not exclusive) record of research in this area over the last decade [15,31,36,77,22].

2 Feature Interaction

In software development a *feature* is a component of additional functionality – additional to the core body of software. Typically, features are added incrementally, at various stages in the lifecycle, usually by different developers. In a traditional telecommunications service, examples of features are a *call forwarding capability*, or *ring back when free*; a user is said to *subscribe* to a feature. Features are usually developed and tested in isolation, or with a particular service. But when *several* features are added to a service, there may be *interactions* (i.e. behavioural modifications) between both the features offered within that service, as well as with features offered in another service. While particular interactions may be benign, in general, interactions can be severely damaging to system development and to user expectations. A benchmark on feature interactions in telephony systems can be found in [26,25]. Major research challenges include how to predict scenarios in which there is potential for an interaction to occur, how to detect that an interaction does indeed occur, and how to resolve an interaction.

Most authors distinguish features and services in the above sense, although some merge the two concepts. For the purposes of this paper, the distinction between feature and service is not significant, what is crucial is the concept of *interaction*.

As examples of interactions, consider the following. If a user who subscribes to *call waiting* (CW) and *call forward when busy* (CFB) is engaged in a call, then what will happen when there is a further incoming call? If the call is forwarded, then the CW feature is clearly compromised, and vice versa. In either case, the user will not have their expectations met.

More subtle interactions can occur when more than one user is involved. For example, consider the scenario where user A subscribes to originating call screening (OCS), with user C on the screening list, and user B subscribes to CFB to user C. If A calls B, and the call is forwarded to C, as prescribed by B's feature CFB, then A's feature OCS is compromised. Clearly, if the call is not forwarded, then the CFB feature is compromised. These kind of interactions can be very difficult to detect (and resolve), particularly since different features may be activated at different stages of the call cycle, and indeed at different locations both outside and within the network.

Interactions may occur at any point during a service. This means that with the recent growth in the number of features and services, there is a combinatorial explosion in the number of scenarios with the potential for an interaction. In general, neither manual inspection, nor simple testing, offer tractable solutions. More effective approaches addressing the special requirements of this domain are needed.

The scale and nature of telecommunications services make them prone for interactions. However, the problems exist in other domains, such as computer aided design [100] and process planning [61]. While we focus on telecommunications networks in this paper, we note that this work also has applications in those domains.

There are three major research trends: software engineering approaches, formal methods, and on-line techniques. Software engineering and formal method approaches are also sometimes referred to as off-line techniques. Off-line means that the approach is applied during design-time of features, in contrast to on-line approaches which are applied while the features are actually running.

3 Overview of Paper

The first half of the paper concentrates on the state of the art and the three major research areas; software engineering, formal methods and on-line techniques. For each of these, in sections 4, 5 and 6, respectively, we give an overview of the approach and the motivation for it, how it is employed, and the contributions and difficulties that have arisen as a result. In section 7 we discuss the overall contribution and impact of these streams of research.

In the second half of the paper we look to the future, beginning in section 8 with a discussion of emerging architectures and feature interaction in that context. In the following section we present a considered forecast about how research in feature interactions needs to readjust to address the new challenges posed by the emerging architectures. We postulate several open questions. Finally, we present our concluding summary in section 10.

4 Service and Software Engineering

Service engineering concerns the creation of services. As such it includes specification, development, testing, and deployment of services. Thus service creation can be adapted to help remove feature interactions before deployment. Yet, service creation implies more. It also includes service management in an operational or deployment sense, with an enormous "service surround" covering aspects such as billing, customer records, and fault management. Despite this potential breadth, most feature interaction research in this area has focused on the pre-deployment stages of service creation. An exception is the contribution of Georgatsos, Nauta and Velthuijsen [45] who discuss the role of the Telecommunications Management Network (TMN) over all the management areas, that is, fault management, configuration management, accounting, performance, and security. They argue that TMN can operate in a complementary manner to address feature interactions. However, as this is an exception, we focus on the early stages of service engineering; indeed in practice we focus on the software engineering aspects.

Historically, service creation has been an exercise in software development. So, it is natural that developments and lessons from the wider arena of software engineering have been studied in both service creation and feature interaction. Indeed, service development process models borrow heavily from their software development counterparts. Software engineering is of course a rather general term, but here we use it to refer to development models *and* techniques employed within a development model at one or more stages. In particular, we are interested in models and techniques which address feature interaction.

Clearly formal approaches can also be part of a development cycle, and so in this section we include formal methods where they are used to add rigour to the service creation process. Typically this involves the introduction of a particular notation from elsewhere. However we deliberately exclude formal methods that employ a degree of reasoning to detect feature interactions. This is a large and active body of work worthy of separate consideration in section 5.

4.1 Why use software engineering?

Services in telecommunications, from a programming perspective, are a challenging domain. They are large, real-time, and subject to frequent change (albeit with long development times), yet must exhibit high reliability. This is exasperated by the distributed nature of both the control and the data, both *within*, and *surrounding*, modern communications networks.

We identify two ways that software engineering can aid in feature interaction:

- Indirectly. Software engineering has proved effective for large scale software production. It is ubiquitous in modern software development, albeit that the formality of the processes vary. By building upon this rigour, and using software engineering techniques within the service creation process the potential for constructively handling feature interaction is greatly enhanced. In essence, the rigour that is introduced mitigates against having unanticipated feature interaction. While perhaps not detecting or resolving feature interaction *directly*, the feature interaction literature has embraced this *indirect* approach.
- Directly. It is natural that an industry used to employing software development techniques to handle specific aspects, such as testing, will also want to introduce feature interaction stages within their existing service creation processes. In this case, a particular method, notation, or technique is applied *directly* to tackle feature interaction detection, resolution, or avoidance.

4.2 How are software engineering techniques used?

We characterise applications of software engineering into two major approaches:

- Focussed techniques. Here, a particular technique used elsewhere in software engineering is introduced into the service creation process with a major goal being the elimination of interactions. These are often leading edge techniques, attracting interest in the more general software engineering area. Typically, specific notations (defined for other purposes) are introduced.

- Process models. In these processes, software development models are adapted to the service creation role, again with the aim of eliminating feature interactions. These approaches tend to define a complete development process, or at least a significant part of it. Here, the emphasis is on detecting feature interactions at an early stage in the life cycle.

We discuss each of these in more detail.

4.3 Focussed Techniques

Reference	*Introduced technique*	*Notation*	*Service Creation Stage*	*Approach*	*Experimental Results*
Braithwaite, Atlee [16]	Layered State Transition System	State Machines	Specification	Detection	case studies
Bredereke [17]	Program families	CSP-OZ	Requirements	Filtering	case studies
Hay, Atlee [55]	Feature Composition	Labelled Transition System	Design	Avoid	no
Heisel, Souquieres [56]	Requirements elicitation	agendas with system state traces	Requirements	Filtering	no
Iraqi, Erradi [63]	Composition of FSA	MONDEL	Specification		case studies
Keck [68]	IN topological relationships	IN-style BCMs	Specification	Filtering	case studies
Kelly, Crowther, King, Masson, DeLapeyre [70]	SDL	SDL	Specification modelling	Indirectly detect	case studies
Kimbler, Kuisch, Muller [78]	Filtering	Interaction Tables	Requirements	Filtering	yes: industrial scale
Kimbler [74]; Kimbler, Sobirk [79]	Use Case Driven Analysis	Service Usage Models and Use Case Model	Requirements	Filtering	case studies
van der Linden [123]	Separation and Substitutability	-	Design		no

Reference	*Introduced technique*	*Notation*	*Service Creation Stage*	*Approach*	*Experimental Results*
Nakamura, Kikuno, Hassine, Logrippo [95]; Amyot, Charfi, Corse, Gray, Logrippo, Sincennes, Stepien, Ware [5]	Use Case Maps	UCMs, LOTOS	Requirements, Design	Filtering, Detection	yes; industrial scale
Prehofer [104]	Feature Oriented Programming	JAVA	Modelling Requirements	Detection	case studies
Turner [116]	Formal version of CHISEL	CHISEL, LOTOS, SDL	Specification, Design	Indirectly avoid	no
Turner [117]	Feature oriented architecture	ANISE, ANGEN, ANTEST	Specification and Test	Indirectly avoid	no
Utas [122]	Pattern languages	Communicating FSM	Implementation	Avoid	yes; industrial scale
Zave, Jackson [130,132,131,64]	Feature Architecture	Diagrammatic	Design	Avoid	yes: industrial scale
Zibman, Woolf, O'Reilly, Strickland, Willis, Visser [133]	Architecture and process models	-	Design	Avoid	case studies

Table 1: Specific software engineering approaches

Often a service creation or development process is assumed and then specific techniques from software engineering are employed within the development process to detect and eliminate feature interactions. Table 4.3 summarises this body of work. The table describes the technique or notation that the work *introduces*, the notations (or at least the form of descriptions) that are used, and the stages in the service creation process where the technique can be employed. The final column captures the degree of experimentation reported in the relevant papers. This ranges from *no*, through *case studies*, to *yes: industrial scale.*

There is limited overlap within this body of work. A broad range of approaches have been tried, although many originate from the object-oriented software engineering community. It is difficult to find a strong common thread of approaches across different research teams, each team would appear to follow their own area. In other words, there is limited consensus of approach. However, two trends are discernable. The first trend as identified by Hay and Atlee [55], is to provide a service architecture that constrains designers to provide "safe" arrangement of features. This includes the work of Zave and Jackson on DFC [130,132,131,64], van der Linden [123], Utas [122], Zibmann et.al. [133], Braithwaite and Atlee [16], Hay and Atlee [55], and to a lesser extend Turner [117]. The second trend is the value placed upon filtering. A filtering stage is seen as a cost effective way of handling the large number of potential interactions by removing unlikely combination of features and services, hence only likely candidates are considered at the detection stage – a costly and time consuming activity. A number of authors view their approach to be suitable as a filter, and so reduce the number of cases that need to be considered.

4.4 Process models

Two process models have been proposed that employ specific stages to handle feature interaction. The first approach [29,28] was developed by Bellcore (now Telcordia) and is aimed at an AIN environment. It describes in some detail the description and specification tasks, with the latter supporting the feature interaction analysis. Both network and user requirements are used to generate services and use is made of both public and proprietary notations and tools. The feature interaction analysis stage is carried out using a proprietary method with proprietary tool support.

The second approach [75] assumes an existing service life cycle and augments it with a separate process specifically for handling interactions. This process, called Service Interaction Handling Process, or SIHP, was developed by the P509 EURESCOM project. SIHP introduces feature interaction filtering, feature interaction detection, and finally a feature interaction solution stage. It does not require or proscribe any particular techniques, however it does require a change to the associated business model. More generally, the P509 project took a very broad look at feature interaction, for example the TMN work cited earlier [45], was carried out within the auspices of this project.

4.5 Contribution and Difficulties

Software engineering techniques offer the potential of a strong, industrial scale approach to feature interaction. The difficulty is assessing the effectiveness of this body of work within the context of feature interaction. Intuitively, software engineering has proved helpful, but it is unclear to what extent this applies explicitly to the field of feature interaction. The final column in Table 4.3 shows the degree of experimentation carried out for focussed approaches. We would argue that there is insufficient evidence published to draw strong conclusions; moreover the papers on processes for feature interaction do not report any experimental results.

While it is tempting to suggest that it is likely that many companies will follow the software engineering route for large scale approaches, we are struck by the limited number of *industrial scale* papers in this area, indeed many of the contributions are from purely academic sources. Clear exceptions are the papers [5,122,78,58,64] which report industrial scale studies. Filtering offers a more coherent picture, and while there is diversity of approach, it would appear to be the preferred framework. In other words, there is an acceptance by many that automated approaches are only suitable for directing a comprehensive manual analysis.

We now turn our attention to the second major research trend, formal methods.

5 Formal Methods

Formal methods have been employed in a variety of ways to analyse feature interactions. By the term “formal methods”, we mean the application of a wide range of formal description, modelling and reasoning techniques. These techniques include, for example, classical, constructive, modal and non-monotonic logics, process algebras, finite and infinite state automata, extended automata, petri-nets, transition systems and languages such as SDL, Promela, Z, and LOTOS.

Formal methods have been applied primarily to the problem of detecting *service level* interactions, that is interactions that are independent of an actual implementation. Thus, the focus has been mainly on interactions between feature requirements and/or specifications. While all of the formal approaches are invariably off-line, they need not necessarily occur at design-time, prior to implementation; in many cases specifications are re-engineered or derived from observations of an implemented system. In this regard, the application of formal methods in this domain is rather more liberal and creative than in other domains.

5.1 Why use formal methods?

There are a number of compelling arguments for employing a formal approach to feature interaction detection:

- A formal description or model forces assumptions and contexts to be made explicit; incorrect or ambiguous assumptions are often a source of interactions.
- Operational systems are notoriously poorly documented. Interrogation of a formal description or model may provide the only answer to a query about behaviour, short of actually trying out a query on a live system or testbed.
- Formal descriptions may make it easier to define classes of interactions in a more abstract way.
- Automated analysis and reasoning techniques are usually applicable; there have been great advances in automated reasoning tools within the last few years.

Formal methods have been used to both validate expected interactions and detect unpredicted ones. While clearly the latter is the ultimate goal, the former is necessary to show proof of concept; it is probably fair to say that the majority of published results are still at the proof of concept stage.

5.2 How are formal methods used?

We characterise applications of formal methods by three major approaches:

- Properties. In this approach abstract *properties* of features and the basic service are defined, usually in a logic. Interaction is expressed in terms of that logic, usually as inconsistency or unsatisfiability.

- Behaviour. In this approach a *behavioural* description of features and the basic service is defined. These descriptions have an "operational" feel to them, usually incorporating state and or temporal aspects, and they are expressed in terms of variants of automata and transition systems. Interaction is expressed in a variety of generic ways such as deadlock, nondeterminism, etc.
- Properties and Behaviour Model. In this approach, features and the basic service are defined by both behavioural and abstract properties. The former is expected to be a model for the latter. Interactions occur when features (plus basic service) satisfy their respective properties individually, but when they are combined, the conjoined properties are not satisfied.

We note that other classifications of formal methods approaches have been restricted to two: our first approach is sometimes referred to as *satisfiability* and our third as *satisfaction on a model*. We believe that it is important to distinguish another approach, *behaviour* only, as it raises numerous interesting ways to define interaction.

We give a brief overview of each of the approaches below. The overview is not meant to be exhaustive, but rather gives a flavour of each approach with representative papers. We have not tried to indicate the relative contribution of each of the different citations and note that in some cases, several different aspects of feature detection are described within one paper (each being characterised by a different approach). We have usually picked the most significant aspect, though occasionally we have mentioned more than one aspect (e.g. in [53] and [111]).

5.3 Properties Only

A number of property only approaches are summarised in Table 2. Acronyms for the logics are: TLA (temporal logic of actions), LTL (linear temporal logic), FOL (first order logic), PL (propositional logic), CTL (computation tree logic). In nearly all cases, automated reasoning is used: theorem proving or model checking, as appropriate. For example, the theorem prover PVS is employed in [48] and the model checker COSPAN is employed in [40]. In some cases there is also a bespoke specification (or requirements) language which is (usually automatically) translated into the logic, prior to interaction analysis.

Reference	*Property Language*	*Interaction Detection*
Gammelgaard, Kristensen [44]	transition rules	realisability
Blom, Jonsson, Kempe [13]	TLA	inconsistency/deadlock
Bostrom, Engstedt [14]	DELPHI (PL)	inconsistency
Frappier, Mili, Desharnais [41]	FOL + relations	inconsistency
Rochefort, Hoover [106]	Constructive Logic	satisfiability
Gibson [48]	FOL & TLA	invariant violation, nondeterminism
Felty, Namjoshi [40]	LTL	inconsistency

Table 2
Interaction Detection – Formal Description of Properties Only

5.4 Behaviour Only

A number of behaviour only approaches are summarised in Table 3.

Reference	*Behaviour Language*	*Interaction Detection*
Thomas [111]	LOTOS	nondeterminism
Bergstra, Bouma [8]	Interworking (synchronous MSCs)	merge-inconsistency
Blom [12]	MSC variant	inconsistent post-conditions inconsistent event "allowedness"
Khoumsi [72]	Extended FSA	deadlock, livelock, nondeterministic uncontrollable or conflicting state
Thistle, Malhame, Hoang [110]	Control theory	conflicting languages
Au, Atlee [7]	State Transition Machines	control/data modification, resource contention, unreachable states
Nakamura, Kakuda, Kikuno [94]	Finite State Machines	deadlock, loops, nondeterminism
Yoneda, Ohta [129]	State Transition Rules	abnormal state/transition disappearance of normal state/transition
Hall [53]	State Transition Diagrams	inconsistent state changes, inconsistent observable actions
Bruns, Mataga, Sutherland [19]	Chisel variant/ML	order sensitive, nondeterminism
Plath, Ryan [102]	CSP	deadlocks
Accoursi, Areces, Bouma, de Rijke [2]	Constraint Programming	model existence
Khoumsi, Bevelo [73]	Extended Finite State Automata	nontermination, variable inconsistency
LaPorta, Lee, Lin, Yannakakis [87]	Finite State Automata	language difference

Table 3
Interaction Detection – Formal Description of Behaviour Only

In these approaches there is a wide variety of ways in which to define interaction, though most can be expressed as some form of analysis for reachability, termination, deadlock, nondeterminism, or consistency. In many cases, automated tools have been developed for the analysis, or when a standard language is employed, then the associated tools are used. For example, in [103] the model checker FDR is employed to find deadlocks in CSP specifications.

We note that there are a number of approaches, loosely known as "pragmatic" and not explicitly using a formal approach, that could be understood as having a formally based justification – though the authors have not made the explicit connection. For example, the CCM-FCA (call-context model, feature context assumptions) approach of [99] is based upon features as sets of tuples. Interactions are detected by various classes of conflicts, which could be understood as various types of overlaps between tuples, or incompatibilities. However, since this is not made explicit, this work is not considered here.

5.5 Properties and Behaviour

A number of approaches which employ property *and* behaviour descriptions are summarised in Table 4. MSC is an abbreviation for message sequence charts.

Reference	*Property Language*	*Behaviour Language*	*Automated Reasoning*
Combes, Pickin [32]	LTL	SDL	none
Stepien, Logrippo [108]	LOTOS	LOTOS	simulation
Thomas [111]	μ-calculus	LOTOS	restricted model-checking
Gibson [47]	TLA	LOTOS	none
Capellmann, Combes, Petterson, Renard, Ruiz [30]	MSC	SDL	simulation
Kamoun, Logrippo [66]	CTL	LOTOS	goal oriented trace execution
Hall [53]	Symbolic transition predicates	P-EBF	symbolic transition checking
Plath, Ryan [101]	CTL	SMV	restricted model-checking
Bousquet, Ouabdesselam, Richier, Zuanon [34]	Lustre	linear past temporal logic	random tests
Calder, Miller [24]	LTL	Promela	full model-checking

Table 4
Interaction Detection – Formal Description of Properties and Behaviour

In all cases, the basic concept of interaction is the same. If feature F_1 *satisfies* a property ϕ_1, written $F_1 \models \phi_1$, and $F_2 \models \phi_2$, then when the two features are combined, denoted $F_1 \oplus F_2$, does, $F_1 \oplus F_2 \models \phi_1 \wedge \phi_2$? If the answer is no, then there is an interaction. A key aspect is how $F_1 \oplus F_2 \models \phi_1 \wedge \phi_2$ is verified (is it even decidable?). With the exclusion of [53], automated reasoning is with standard, generic tools. However, in nearly all cases, there are problems with combinatorial state-space explosion. Only a few approaches (e.g. [24]) are able to perform exhaustive verification, and so many rely on examination of selected, or incomplete behaviours only.

5.6 Contribution and Difficulties

The potential contributions of employing formal methods (as described in 5.1) are clearly borne out by the results obtained in the various applications mentioned above. For example, nearly all employ automated analysis tools; significantly, these are not on the whole bespoke tools but often industrial strength, generic, reasoning tools. Many of the authors of (positive) detection results comment on how formality has forced them to make explicit aspects of features that have previously been ambiguous. In the words of Hoover and Rochefort [106], *initial "accidental" correctness is a major cause of feature interactions.*

The completeness of any formal approach is an important issue, depending on *how* feature interaction and the features themselves are defined. The properties and behaviour approach depends upon the details of the properties defined and the behaviour only approach depends upon the way in which interaction has been defined. Possibly the latter approach is more suited to detecting unpredicted interactions, because interaction is defined by generic criteria. Further, when modelling features for interaction detection, getting the right level of detail and abstraction for feature description is difficult, and has consequences for detection analysis. For example, under-specification introduces the problems of the *frame* problem (i.e. what is *not* changed by a feature), whereas over-implementation may introduce false positives.

A common difficult issue is compositionality, specifically non-monotonicity between new features and the original service. The crux is that the addition of new features nearly always requires a modification of the original service. For many formal approaches, this presents quite a problem. Velthuijsen [126] first articulated the problem and suggested non-monotonic logics as a solution. Most researchers still remain within monotonic frameworks, but have introduced special techniques to cope with the non-monotonicity. A notable contribution is Hall's proposal of foreground/background models [53]. Using this distinction, he is able to add new features that "perturb" the basic service (i.e. they are non-monotonic extensions) and most importantly, is able to rule out many spurious interactions which can now be understood as the result of interactions between the background behaviour of one feature with the foreground behaviour of another feature. This is a good example of a formal approach helping to explain a difficult and previously poorly understood phenomena.

State space explosion remains a challenge, but various approaches exploiting symmetry such as [94] or reduction techniques such as [24], are very promising.

Some of these difficulties are not unique to formal approaches, but are shared with other approaches. If they are overcome, in this context, what is the contribution of formal approaches, in particular? The evidence clearly demonstrates that they force assumptions and contexts to be made explicit, provide a rigorous yet practical way to define classes of interactions, and lend themselves to automated analysis and reasoning techniques, usually by standardised tools. However, at the end of the day, the results concern the formal model – they are only of value if they can be properly related back to the operational world and the software development process.

Formal methods are almost exclusively an off-line technique for dealing with feature interactions, we now turn our attention to on-line techniques.

6 On-Line Techniques

On-line techniques are intended to be applied at service runtime in a network. Here we consider both live networks and test environments within laboratories, such as captive offices. Usually on-line techniques provide a combination of detection and resolution mechanisms as on-line detection is only productive if the problem can be resolved at run-time.

6.1 Why use On-line Techniques?

Service engineering and formal methods present a number of drawbacks, many of which have the potential to be overcome by using on-line techniques. The following list discusses some major points.

- On-line techniques are applied to an *active system* in its *natural environment*. This avoids the problem of formal method approaches which operate on a model of the real system.
- Support for a *quick time-to-market* for new features. Extensive testing of all possible scenarios is not necessary.
- On-line techniques are more *future-proof*, as they are part of the system and not the development process. This allows to automatically include all newly deployed services into the detection and resolution process.
- An increasingly *multi-vendor market* removes global knowledge about services in the network – and the possibility to change those supplied by others. In this context, on-line techniques provide the only way to manage interactions.
- Feature interactions can be detected and resolved without *tampering with legacy systems*. Formal method approaches fall short here as system documentation and thus the required detailed information is often not available.

While some on-line methods have been implemented into commercial products (e.g. Mitel MediaPath [20]), most on-line techniques have been applied in test-beds (e.g. Touring Machine [6], Desk [93]). This might be due to mainly two reasons: some on-line techniques use detailed per-service design-time information, such as resolution tables, while others require changes to the network architecture to allow inter-feature communication. Clearly, these pre-requisites have an impact on all four of the possible strengths listed above. A number of on-line approaches have been developed so far. In the following, we discuss characteristics of approaches and highlight their strengths and weaknesses.

A major disadvantage is the processing overhead that is introduced into the real-time system. Clearly this is of particular concern on a live network.

6.2 How are online techniques used?

On-line techniques can be grouped in a number of ways. Probably the most important ones are by the location of control and by the required type of information.

Two classes of approaches can be identified considering the location of control:

- *Feature Manager* based approaches. An entity, usually called the feature manager, is introduced into the network with the capability of observing and controlling the call processes. Hence, the control of the call is located with the feature manager. So far mainly centralised approaches featuring a single feature manager have been developed. However, distributed architectures for managers are also possible.
- *Negotiation* based approaches. Individual features have the capability of communicating their intentions to each other and negotiating an acceptable resolution. Most approaches advertise a direct communication where the call control resides with the features. However, if no resolution is possible the conflict can be forwarded to a third party to resolve.

Three groups of approaches can be identified considering how the required information is collected. This categorisation is orthogonal to the previous one:

- *A-priori information* Approaches belonging to this group are applicable at runtime, but they make use of data collected at design-time. This information can be in two forms, as a per service information or as a matrix relating all services. Clearly, information on a per-service base is preferable to information per service pair. With the growing number of services the amount of information grows linearly for the per-service case but exponentially for the per service-pair case.
- *Isolated on-line environment* The information for detection and resolution is gathered at runtime. However, not in the live system but in a closed environment where special requirements can be met. These include the presence (or absence) of other features. Here the issues of a poor run-time performance are eased.
- *During run-time* These approaches do not require any special arrangements. The required information is collected during run-time of the feature in the target environment.

The way information is gathered is crucial for the success of on-line techniques. Collecting information at run-time is the major advantage over formal methods and software engineering approaches. Table 5 provides an overview of approaches.

6.3 Feature Manager – A-priori Information Approaches

A basic feature manager is defined by the ITU-T standard for Intelligent Networks [119], however this only prevents multiple instances of IN and non-IN services being active in the same call segment. More advanced approaches use feature managers to detect and resolve interactions, by using information about the features and their potential interactions to allow multiple non-interacting features to be active across the call (in one or more call segments).

Homayoon and Sing [60] propose such an approach, whereby the feature manager is provided with a number of tables describing relations between two features. The status of one of the features is examined and then the activation or use of the second is allowed or disallowed. Similarly, [21] proposes a feature manager that only passes an event to features that are known to be non-interacting based on tables. Activation is not considered.

Reference	Control	Information acquired at	Experimental Results
Marples, Magill [91]	Feature Manager	run time	case study, testbed
Homayoon, Singh [60]	Feature Manager	a-priori	
Cain [21]	Feature Manager	a-priori	
Fritsche [42]	Feature Manager	a-priori	
Reiff [105]	Feature Manager	run time, general rules	case study
Tsang, Magill [112,113]	Feature Manager	captive environment	case study, testbed
Aggoun, Combes [3]	Feature Manager	captive environment	
Velthuijsen [125], Griffeth, Velthuijsen [51]	Negotiating Agents	a-priori	case study, industrial
Buhr, Amyot, Elammari, Quesnel, Gray, Mankovski[20]	Negotiating Agents	a-priori	
Amer, Karmouch, Gray, Mankovskii [4]	Negotiating Agents	a-priori	case study, industrial

Table 5
On-line Approaches – Detection and Resolution

Fritsche [42] determines at runtime which features are “interested” in a proposed event. A specification of the features is provided to the feature manager in form of roles, i.e. a feature’s impact on a device. Features apply changes to devices and the feature manager observes whether or not roles are violated (an interaction). Interactions thus found are resolved by a predefined resolution matrix.

While these approaches extend the ITU-T IN feature manager, they are limited in their applicability to expanding systems. This is because the feature manager needs to be provided with (detailed) information on new features. Clearly, this is a problem if the features are provided by multiple businesses. This is even worse for the case where information on feature pairs are required. Furthermore, legacy systems are very hard to handle by these approaches. The information required by the feature managers is often simply not available.

6.4 Feature Manager – Captive Environment Approaches

The approaches described above require a-priori data about features and their potential interactions. Since this is a serious drawback with on-line approaches, approaches which circumvent that requirement have been developed. One way of achieving this is to introduce, during run-time, a separate phase of ”collecting” information before the actual operation of the approach. During this phase the behaviour of the feature is observed and the information is stored in a database.

Aggoun and Combes [3] propose a “pre-deployment” phase where a passive observer gathers information about the feature behaviour in the network. The gathered information is then used by the active observer (essentially a feature manager) in the operation phase of the service to detect and resolve interactions.

Similarly, Tsang and Magill [113] gather behaviour “signatures” of features in an isolated on-line environment (with just the feature under observation being active) and store those in a database. The feature manager then accesses this database during live network operation to detect and resolve interactions.

Approaches in this group are also able to deal with legacy systems. This is because the behaviour of the existing system including the legacy functionality can be seen as a feature. The approach can gather information on the behaviour of the whole legacy system. However, a potential disadvantage of these techniques is that the behaviour collected during the first phase needs to be stored centrally in the network, accessible by the Feature Manager. With an increasing number of features in the network it is crucial that the stored information in kept to a minimum. Otherwise there is a scalability issue.

6.5 Feature Manager – Approaches without restrictions

This class of approaches offers the most flexible way of detecting interactions at runtime. No special information is required and no ”warm-up” phase is needed. However, there are very few approaches known so far. One likely reason is that these approaches are the hardest to develop. The only information the feature manager can collect are the messages sent in the network. Clearly, these approaches can handle legacy systems and also systems with a continuously growing number of features.

One such approach was devised by Marples and Magill [91]. The feature manager assumes an interaction to have occurred when more than one feature attempts to handle an event. They then use a rollback and commit algorithm to determine possible resolutions at runtime; their resolution mechanism is a simple precedence scheme. This approach has been implemented in the DESK testbed [93,92]. Ongoing work by Calder et al. [23] and Reiff [105] extends this resolution mechanism to apply more sophisticated techniques based on general, feature independent, rules.

Approaches in this category have the potential to use the strengths of on-line techniques to the full.

6.6 Negotiation – A-priori Information Approaches

A markedly different alternative is provided by negotiation approaches. Here, features and resources are represented by agents able to communicate with each other to negotiate on their goals, successful negotiation means that an interaction has been resolved. The a-priori information required is hidden in the concept of successful negotiation – it must be known when goals interfere and when they do not.

In an early paper [125], Velthuijsen evaluated a number of distributed artificial intelligence techniques (DAI) to help resolve the feature interaction problem, several approaches have since been developed using the techniques. For example, Griffeth and Velthuijsen use negotiating agents to detect and resolve interactions in [51]. A resolution is a goal acceptable by all parties, and is achieved by exchanging proposal and counter-proposals amongst the agents. Different methods for negotiation have been envisioned: direct (agents negotiate directly without a negotiator), indirect (a dedicated negotiator controls the negotiation and can propose solutions based on past experience) and arbitrated (an arbitrator takes the scripts of the agents and has sole responsibility to find a solution). They concentrate on indirect negotiation. The approach has been implemented in the experimental platform “Touring Machine” [6], although no conclusive report about the success is provided.

Rather than using direct negotiation, Buhr et al. [20] make use of a blackboard. Features are represented by agents which exchange information by writing to a public data space. Other agents can change the information written to the blackboard and a common goal can be negotiated. The success of the technique is reflected in its use in Mitel’s MediaPath product. Amer et al. [4] also use the blackboard technique, but extend their agents to make use of fuzzy policies. Agents set truth-values (0 to 100) to express the desirability of certain goals. These values are then adapted as the call progresses, depending on the values of other agents. In the case of a conflict an event with the highest truth-value is executed.

Negotiation approaches have a huge potential flexibility. However, in the existing network architecture, feature to feature communication is very difficult. Thus these approaches often require a change in the network architectures, e.g. by introducing agent components. Furthermore, often a negotiation hierarchy is needed to guide the negotiation.

6.7 Contributions and Difficulties

On-line techniques offer strengths which no other technique possesses. They address both detection and resolution. While this is not often the case with other approaches (e.g. formal methods), automated resolution is especially critical with on-line techniques. At design time, manually changing specifications is sufficient. However, feature interactions detected at run-time need to be resolved instantaneously to keep the integrity of the network. On the other hand, work reported so far suggests that run-time resolution is difficult. This is due, in part,to the limited amount of information available. Making an informed decision without much knowledge on the features is hard. Not surprisingly, approaches requiring a-priori knowledge offer the best resolutions.

However, the biggest potential of on-line techniques lies in their ability to cope with additional services in the network at runtime and thus allowing for open and expandable systems. A-priori knowledge is a major obstacle to use this potential. Thus, in terms of feature interaction resolution, many approaches opted for the obvious choice: terminating the call. However, is this a ”bad” resolution? A single call is very cheap. Thus terminating a call might at most annoy the affected users. On the other hand a complex algorithm finding a good resolution is possibly very expensive in terms of its run-time behaviour. Further, the resolution might still confuse the users and mismatch their expectations. To progress work on this issue work is ongoing to augment run-time approaches with feature independent information from off-line techniques to guide the resolution. These approaches are referred to as hybrids. There is very little published work on hybrid approaches to date, however, two examples are the work by Calder et al. [23,105] and by Aggoun and Combes [3].

One drawback of run-time techniques is the strong link to the applied network architecture. If the network changes the on-line techniques need to be adapted to the change. Off-line methods may profit from their abstraction and may not require major adjustments.

Many run-time approaches use centralised control and centralised data which is hard to achieve in current networks. However, this is not a inherent problem with run-time approaches but rather with constrains due to the limited signalling in today's networks and the state of research. Few approaches which do not require central control or data have been developed (e.g. [51]) but appear to have been difficult to implement.

In summary, no conclusive study of industrial usage is available. The Touring Machine and Media Path products appear to be the only industrial implementations. The feature manager approaches are mainly academic prototypes, with the Desk testbed being the closest to a real system implementation.

Why on-line techniques are not being used more widely, despite appearing very promising, cannot be judged here. We can only speculate that the feature interaction problems encountered so far could be handled in a pragmatic fashion at design time. The required architectural changes have proven to be too costly in the past.

7 Overall Contributions of Previous Work

A large number of contributions have been made by both academia and industry over the past decade in order to solve, or at least advance, the handling of the feature interaction problem. The major streams described above have evolved, each offering their own benefits, but also influencing the others.

Service engineering is influenced by software engineering, and offers an attempt to adapt general practice to help in the development of services. On the other hand, formal methods have been used extensively to help finding solutions. Enhanced system "architectures" have been developed for run-time solutions.

An obvious question is: *Have we made any significant advances in solving the feature interaction problem?* The software processes are mostly academic and case studies have rarely been performed. There are few operational systems using the run-time approaches. Formal methods seem promising but have not yet delivered any clear advances. Again, most case studies of formal method and run-time approaches include only a very limited set of experiments. (An exception is Holzmann and Smith's use of

SPIN to verify parts of the Lucent PathStar access server [59,57,58].) Furthermore, the tests are performed on known interaction cases. New interactions, as the ultimate goal of the research, have not been the focus. So, the pessimistic answer to this question is that the problem is still unresolved and we have failed thus far.

However, this is not the true picture, which is far more complex. While the work using formal methods has not yielded a clear solution, it has greatly advanced the understanding of the problem. It has highlighted the fact adding features to a service is fundamentally non-monotonic and potential approaches must address this. A range of new tools and notations has been developed or advanced, thus the repertoire of powerful methods increases rapidly and should enable quicker advances in the future. Interestingly, the telecommunications industry has encouraged the use of formal methods, something which can not be said in general, for the uptake of formal methods (except perhaps for safety-critical industries).

Because of an ever growing pressure on the market, industry is understandably reluctant to discuss the methods they use. In this context, it becomes very hard to judge the impact of the feature interaction work, but we assume that certain aspects have indeed been taken up and are used by industry to limit the effect of the feature interaction problem and to reduce the time to market. Furthermore, a number of approaches have competed in two Feature Interaction Contests finding unexpected interactions [50,84,83].

One way to evaluate the contribution is to compare the effort in the three major areas, we found a ratio of roughly 1:2:4 of publications in the areas on-line techniques, software engineering and formal methods, respectively. On this basis it might appear that invest/return relation is worst for the formal methods based approaches, an alternative interpretation is if on-line techniques are so promising, then why is there a dearth of material about them? Indeed, over the past few years, there has been a decline in the number of papers concerning on-line techniques presented at the major conferences. This does seem puzzling.

A major drawback of the current work can be seen in the strong concentration on *POTS* – the plain old telephone service. This concentration is justified by a good understanding of the problem in this domain, coupled with the availability of data and specifications. However, the recent changes in the telecommunications domain towards open markets, integration of voice and data and an ever growing requirement for new functionality, leads to new problems and requires thorough consideration.

The original aim of detecting and resolving feature interactions in telecommunications networks needs to be revised in light of these developments, raising the question: *Are we solving the right problem?* We need to determine how the changes in the telecommunications world influence the validity of the problem, and also must adapt the problem description to reflect those changes. The results from the POTS world can and should be applicable. However, this requires a sound understanding of the emerging technologies and also a wider view of the telecommunications domain , including both the technologies involved and also market situations and business behaviour.

The rate of change is very fast and research into the feature interaction problem is in danger of lagging behind. Consequently, we now turn our attention to emerging architectures.

8 Emerging Architectures

In recent years telecommunications has changed rapidly. There are two drivers for this development: deregulation of the telecommunications market and technological advances. These two forces are tightly coupled: the technological advances would not be possible without the deregulation and similarly the deregulation would not be successful without the technological changes. In the next two sections we discuss the impact these two drivers have on the feature interaction problem. We note that within the context of emerging architectures, the emphasis is on services, rather than on features, per se. Consequently the literature refers to interactions between services; for consistency we maintain that terminology here, noting that the distinction between service and feature is not significant when considering the issues of interaction.

8.1 Deregulation of the Telecommunications Market

Deregulation requires the incumbents to open up their networks to third parties. In other words, third party businesses now have access to the networks in order to develop and provide services. This changes the previously very tightly controlled market into a highly competitive one: offering services is a very lucrative business and is a major source of revenues in the telecommunications market. Beside mass market services, niche markets can also be targeted. Hence a number of independent service providers will enter the market and compete with the traditional operators for a share in the services market [39]. Networks can almost be seen as a mere commodity [118]. Hence, as networks are opened up, a large number of new businesses are entering, or will enter, the market [76]. Interestingly, separating switching and service provisioning to allow service provisioning by independent providers was an original aim of the Intelligent Network. However, the Intelligent Network did not achieve this.

As a consequence of deregulation, a new scale of competition is introduced into the market [37]. This has a direct impact on the number of services offered on the network (it will be much higher than in traditional telecommunications networks). Moreover, because service providers will want to differentiate each other more and more, the functionality and thus the complexity of services will increase dramatically. Additionally, there will be a number of services which are very similar – but not exactly the same. Further, due to the stiff competition, in order to survive, a short time-to-market is essential for the providers [86,85].

As is common in a competitive business environment, businesses do not disclose details about their products. In other words, service providers will keep detailed knowledge about their services private. Hence, service interaction approaches which do not require detailed service knowledge are required. This is even more important considering the huge number of services deployed – it is virtually impossible to collect detailed information on all of them! [80,11]. In fact it will even be impossible to know just about the *existence* of all deployed services, as services will be continuously deployed. So, due to the new structure of the market, a number of current feature interaction approaches may not be not universally applicable anymore. However, in order not to jeopardise the goals of deregulation, such as a competitive multi provider business environment with diverse service portfolios, the feature interaction problem needs to be tackled [86].

Given this context, two basic cases of service interworking (as the base of any service interaction) are distinguished [80]:

- Intra Provider Portfolio (services of the same provider).
- Inter Provider Portfolio (service from different providers).

For the Intra Portfolio case, when the considered services belong to the same service provider, [80] suggest that most of the known classes of approaches can be applied.

However, for the Inter Portfolio case, the requirements for any possible approaches are much harder. This is due to the fact that no detailed service knowledge is available and no a-priori knowledge on the other deployed services can be acquired. Hence, interactions between services belonging to different providers may only be handled at run-time. Moreover, only run-time approaches which do not require detailed service knowledge are applicable [80].

Further, at present a strong move towards convergence of the technologies of traditional telephony networks and the Internet can be witnessed, initial work on identifying related issues for feature interaction management has been carried out. Cameron and Lin [27] compare the two industries:

- Product vs. Service focus. In the latter, the service provider is responsible for the introduction, operation, and maintenance of the services and necessary hardware, in the former, the user is responsible for the introduction, operation, and maintenance of software and hardware.
- Degree of regulation. Regulation in the telecommunications market is much stricter than in the Internet market.
- Amount of competition. The Internet market is much more competitive than the telecommunications market.

As a conclusion, Cameron and Lin [27] state that as the two industries converge, it is likely that the two markets also converge. Thus, the telecommunications market will become less regulated and much more competitive with many more businesses entering and competing for market shares.

8.2 Technological Advances

As mentioned earlier, the changes in the market are both supported and driven by new technological developments. We categorise these developments as:

(1) Development of new Network Architectures/Protocols, such as SIP.
(2) Introduction of a Service Layer with APIs, such as JAIN or Parlay, on top of core networks.
(3) New technologies to provide services in the core networks, such as Active Networks.

All developments have a strong impact on the provided services, including complexity, functionality and interworking. Furthermore, the convergence of the traditional PSTN (Public Switched Telephony Network), the Internet and mobile networks open up completely new ways of communication. A number of researchers have reported on the fundamental changes in new emerging networks [27,76,80,127] and have identified a number of paradigm shifts. We list them here with an indication of the type of development, as given above.

Development	*1*	*2*	*3*
centralised service logic → distributed service logic	x	x	x
functional → object-oriented	x	x	
vertical service provisioning model → horizontal service provisioning		x	x
circuit switched → packet switched	x		x
protocols → Application Programming Interfaces		x	

In the following sections we provide a view on how these paradigm shifts affect service interworking and service interaction handling. Some paradigm shifts are very closely related; this is especially true firstly, for the object-orientation and the use of distributed service logic, and secondly, for the horizontal services provisioning and the use of APIs. Hence, these issues have been grouped into one section.

8.2.1 Object-Oriented Technologies and Distributed Service Logic

In the telephony network, the intelligence is concentrated inside the network and the customer premises equipment (CPE) are dumb telephones. The architecture is based on very few very powerful Service Control Points which run extremely complex and monolithic software. While the small number of processors was meant to help software updates, the structure of the code makes this a difficult endeavour and third party service provisioning impossible [88].

Today there is a strong drive towards packet-based networks. As a consequence control will move from the core of the network to its edges. However, the network still will not be a *dumb* one - only the nature of the provided services will change. Control and intelligence at the edges will increase diversity of the end devices. The network needs to include intelligence to make these incompatible endpoints work together [127]. In other words, while services will increasingly be provided at the periphery of the network, certain core functionality still needs to be provided by the network. Clearly, this leads to distributed service logic, both in the network as well as at its edges. A number of approaches have been applied to achieve this distribution.

Object-oriented technologies have been used in the ROSA project [71] and also by the TINA consortium defining the TINA architecture [62]. TINA introduced the separation of access, service and connection management. Because of this separation, the use of

Distributed Processing Environments (DPEs) becomes possible, and using DPEs such as CORBA, the TINA architecture introduced the concept of distributed service logic. It is widely accepted that future service platforms (e.g. JAIN, Parlay) will be based on DPEs [76].

In such an architecture, many interactions can be avoided. For example, Kolberg and Magill [82] demonstrated how the use of DPEs in TINA eliminates interactions which are traditionally due to limited network support.

Another way by which service logic is provided in a distributed way is the development of Active Networks [109,128]. Active Networks allow for the switches of the networks to execute customised programs on the data flowing through them. In other words, the nodes can perform computations on, and modify the packet contents. Because a packet may be altered in many routers on its way to the destination, active networks greatly increase the distribution of service logic. It is important to note that while Active Networks provide for logic execution inside the network, the information originates at the network edges. This is an essential difference to the traditional networks.

8.2.2 Horizontal Service Provisioning and Application Programming Interfaces

Horizontal service provisioning means that services will no longer be developed for each network architecture separately, but instead, services will be developed in a network independent fashion [27]. In other words, the services will operate on different core networks. This is enabled by the introduction of a service layer which provides basic service capabilities, e.g. call control, billing. Access to these capabilities is via secure and extensible Application Programming Interfaces (API), replacing the restrictive protocols used within traditional telecommunications networks. A number of industrial consortia have adopted the layered approach and develop Network APIs. Major work include JAIN [35,65], Parlay [18,98] and 3GPP's OSA [1] which in turn is based on Parlay. These APIs are being discussed and endorsed by a number of other (standard) consortia, such as the European Telecommunications Standards Institute (ETSI) [38], VASA Forum (formerly the IN Forum) [124], and Softswitch [107].

The xbind architecture [88] is another example of a layered architecture using APIs for communication between the layers. This work was the base for forming the OPEN-SIG group [96] and the IEEE OPENARCH conference series. Subsequently, the IEEE P1520 [97,9] standards initiative for programmable network interfaces was started. An open architecture in network control is being defined and standardised.

Initial studies of the Feature Interaction problem between services deployed on top of Network APIs can be found in [76,80]. Four different categories of service interactions according to their location are identified:

- Interactions between features in the core network, e.g. PSTN, PLMN, H.323.
- Interactions between core network features and service capabilities in the APIs.
- Interactions between service capabilities in the APIs.
- Interactions between third party services offered on top of the APIs.

While the first category contains the interactions known to date, the remaining three classes of interactions are new with the layering of the service architecture as suggested by JAIN and Parlay. The use of APIs and the classes of interactions are depicted in Figure 1.

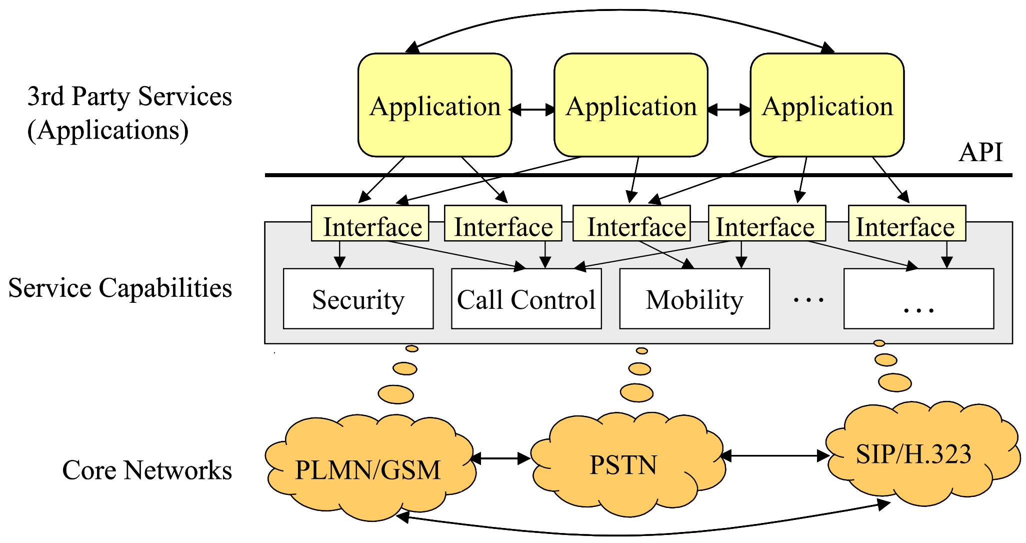

Fig. 1. Potential Interactions and Horizontal Service Provisioning

Active Networks also have a heavy impact on the provisioning of services. Initially they have been conceived to operate inside routers/switches in the network. However, subsequently, work also has addressed application level active networks [43,46]. Here the active code operates on the information exchanged between client and servers, such as WWW browsers and servers, and active code is executed on proxy servers. Hence active networks help to provision services at various layers – from the network layer to the application layer. However, while this helps to increase the distribution of the services, the impact on service interaction handling has not been studied yet.

The new developments do not only mean new classes of interactions but also new opportunities of handling interactions generally. One way of implementing active networks is to use intelligent agents [67]. The close link between Active Networks and intelligent agents might be of interest as a number of feature interaction approaches are based on agent technology. However, no detailed study has been reported so far.

The application of distributed processing environments and of object-oriented technology allow for new ways to control service interactions. An interesting aspect of architectures based on DPEs is that the communications between services is no longer constrained by the network signalling protocol. Instead, information according to the offered interfaces can be exchanged. Thus the signalling bottleneck between services is not an issue anymore. Services can communicate with each other, either directly or via a third component. This means that many signalling limitations experienced in the POTS and IN environments are removed. This also means that services are now aware of each other [80,86] (cf. Figure 2).

This change offers the potential for radically different approaches. In previous architectures features do not communicate: one service is only aware of other implicitly

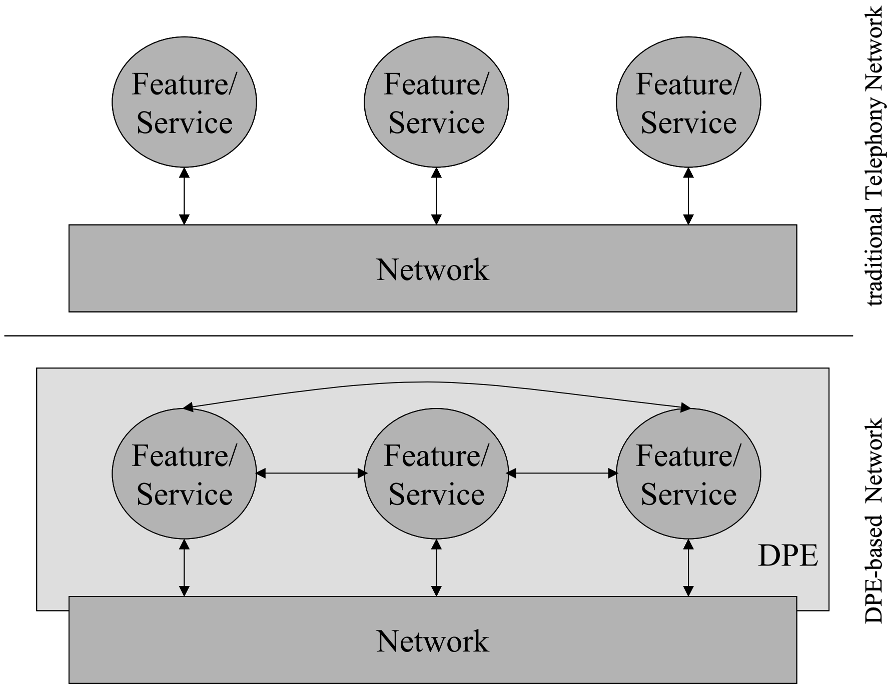

Fig. 2. Service Interworking in DPE-based Architectures.

through the behaviour of the network. However, in new IP-based architectures peer to peer communication between services is possible.

8.2.3 Packet Switched Networks

One of the most important changes in networks is the migration from circuit switched to packet switched networks. There is no longer any doubt that IP will be the ubiquitous transport protocol [27,76]. While the difference between packet and circuit switching is not directly linked to the provisioning of services, it allows for the exchange of multimedia data which is exploited by services.

At present, the work towards Internet telephony has attracted a lot of attention. A number of standards have been developed: SIP [54] by the Internet Engineering Task Force (IETF) and H.323 [120] by the International Telecommunications Union (ITU). MeGaCo (H.248) [33,121] is standardized by both, IETF and ITU. Clearly, the introduction of Internet telephony has a strong impact on feature interaction management. The deployment of new services is much more straightforward - enabled by concepts such as SIP-CGI [90]. Even users themselves may deploy services using languages like the Call Processing Language (CPL) [89].

The type and amount of media carried over the network will change drastically. One major change is the possibility to transmit multimedia data. As a consequence new services handling and managing these data are introduced. Tsang et al [115,114], Zave and Jackson [132] as well as Blair and Pang [11] investigate feature interactions connected with multimedia service, and identify issues connected to bandwidth competition and Quality of Service as crucial.

A major use of active networks is to provide Quality of Service management through mobile agents. While this is still early work it is expected that the work on mobile agents and active networks will impact the work on quality of service and visa versa.

As discussed in the previous section, in packet-switched networks using IP as the network architecture, the signalling bottleneck between services has been removed. Consequently, services are aware of each other and may communicate implicitly or explicitly.

8.3 Feature Interactions in other Emerging Technologies

Feature Interactions are not confined to telecommunications services. In fact, feature interactions are an issue wherever independently developed software components are required to work together (see [10] for a study in component systems). In telecommunications developments such as Parlay, Jain, and also SIP, which allow for a wider participation in the creation and deployment of services, have highlighted the issue. However, feature interactions are also an issue with other emerging technologies outside core telecommunication applications. This drive is supported by technologies such as CORBA, DCOM, and JAVA RMI, which allow for easy communication between distributed components. Importantly, specifying (and even standardising) syntactic interfaces between components (or features, services, agents) using technologies such as CORBA's IDL, or XML, does not prevent feature interactions. What is needed are semantic specifications. However, such specification will be complex and are hard to achieve! Below we list some areas where feature interactions may be a threat.

Web services are software applications whose interfaces are defined in WSDL (Web Service Description Language) – an XML format for describing network services as a set of endpoints operating on messages. *Grid computing* also employs this model, for example the Grid services in OGSA (Open Grid Services Architecture). As discussed above the interface specifications are insufficient and hence feature interactions will be an issue between web services and also between grid services.

Recently, work on *Networked Appliances* has attracted increasing attention. Household (or business) appliances are networked. This allows for remote control using services offered by independent service providers. These services may interwork either directly via a jointly controlled appliance or by appliances affecting each others behaviour. A study has identified several classes of interactions in this domain [81].

Also developments known as *ubiquitous computing* and *ubiquitous services* involve interworking services from a number of different sources. Here people expect to find the same user environment including services wherever they are. This applies to the computing environment (have access to the same applications as at home) and also to the communications system. The issue is the compatibility of services across domains. Using the mobile telephony network, when users travel abroad they expect the same services to be available as in their home network. Also call legs may span across a number of national networks and the services involved need to interwork correctly. Again the potential for feature interaction is given.

Another domain where feature interactions will be an issue are cars. As cars get more intelligent with new features, the amount of software involved is rising fast. A often referred to example is where a thief hits the front of a stationary car and all doors automatically get unlocked. This is because the safety feature of the car assumes an accident has occurred and to allow passengers to escape and paramedics to reach potentially injured passengers all doors get unlocked. This could be seen as the security feature (keep all doors locked) is compromised by the safety feature. Clearly, this is undesired behaviour of a stationary car with no passengers inside.

The commonalities between all these examples are that the services are developed independently by separate businesses (or perhaps even by end-users themselves). Consequently, knowing about the existence of other services or even getting details on their behaviour is often difficult because of the competitive nature of the market or simply because of the number of offered services. Cars appears to be a closed environment at first glance, i.e. only a well understood set of services is used inside a car and hence testing or off-line analysis may be sufficient to eliminate feature interactions. However, this hope may be short-lived as there is a move to connect cars to the Internet. Hence external services may also affect the control of cars.

8.4 Contributions to date

The development and deployment of new network architectures and services is advancing very rapidly, this is supported by the increasing deregulation of the telecommunications market.

Feature interaction research in the area of *Emerging Architectures* is still at a very early state. Work so far has focused on the expected changes and started to look at the likely impact these may cause. While the removal of the signalling bottleneck also removes a large class of interactions, others remain, or are newly introduced. The major significant change is that services may communicate with each other, either directly or indirectly. This is a major departure from the POTS world and consequently new techniques and approaches will be required to deal with interactions in the context. Existing techniques may sometimes still be applicable, as demonstrated by Hall [52] in work on interaction between email services.

In the next section we look ahead and revisit the three classes of approaches discussed earlier in this paper: software engineering, formal methods, and online techniques. More specifically, we identify issues which require to be addressed in the future, and suggest approaches or classes of approaches which might be applicable.

9 Forecast

Feature interaction is becoming a more important issue as the scale and frequency of the problem increases. A number of powerful *drivers* have been noted; the rapidly increasing number of services, a requirement for a shorter time-to-market, a market with multiple vendors employing proprietary services, the possibility of user-created services, and a continuous service deployment.

Arguably the most drastic change in future telecommunications systems is the move towards a competitive business environment. This is cause to a number of open questions. Because the answers will impinge on the nature of future approaches to the feature interaction problem, it is important to answer the questions *before* any new technical feature interaction approach is developed. Some of those questions are:

- Are providers, vendors and operators of Next Generation Networks aware of the impact of feature interactions on their future service offerings? Does the industry care about the problem?
- What will be the impact of the feature interaction problem on the business model of the open services marketplace with a much higher number of third party service providers?
- Who owns the customer? In other words, if there is a problem involving services from a number of service providers, who is responsible for resolving the issue? Possibilities include the providers involved, the vendors supplying the services to the providers, the network operator, and even the users themselves. Clearly, to be able to answer this question the structure of the market needs to be understood.

These issues apply to any new multi-vendor, multi-provider environment, be it SIP, JAIN, or Parlay. Finding an answer to these questions will require understanding the business model and the responsibilities and roles of the entities involved in developing, selling, and maintaining services.

For instance, in a Parlay environment, if the network operator is responsible to resolve the interactions, a likely place for an approach is inside the API framework interface, or if the service providers are responsible, the services themselves might need to contain feature interaction handling functionality. Clearly, the location of the handling will affect the types of approaches required. Further research is needed to find answers to these open questions. Moreover, the role of the regulator and standardization need to be defined.

Another major issue in the future will be the compatibility of services associated with the issue of ubiquitous computing. Users will expect that they have the same services available whatever handset they may use and also wherever they are. For instance when travelling to a foreign country, users will expect that they still have the same services available as at home and that the services operate in exactly the same way. Furthermore, users will expect services to work across networks. Conference calls may span multiple networks, for instance with call parties in USA, France and Japan.

In addition, the *nature* of feature interaction is changing as the networks which support the services are changing. Research in this area needs to readjust and address the new challenges and opportunities. We believe that these issues are affected by four major developments. While all four set new challenges, it is important to note that the first two also offer fresh opportunities.

- Richer signalling. This removes the signalling bottleneck and entire classes of interactions which dominated the POTS world are simply avoided.
- Communication between services. This is now made possible through (increasingly) standardized middleware and API's and raises the questions of what (and how) do services need to "know" about each other to detect and resolve interactions, and who or what mediates?
- Performance in IP networks. Quality of service is now an issue, not only for traditional POTS-like services, but interactions between multi-media services can compromise quality of service guarantees.
- Billing. While it was always clear that billing interactions were occurring in the POTS world, they were generally considered to be a minor concern and not worth fixing. However, billing in a multi-vendor environment, with the separation of service and network providers, is much more important and has real financial consequences. These are made more acute by the move towards run-time, electronic bonding, as opposed to the traditional off-line post-call processing for billing in the POTS world.

Consequently, research priorities must be to

- Determine the minimum information that services need to communicate to other services in order to detect and resolve interactions.
- Develop agreed interfaces between services for the exchange of information.
- Develop quality of service measures from a service perspective (rather than a network perspective, as at present).
- Incorporate performance as a part of the functionality of a service/feature.
- Incorporate billing as a part of the functionality of a service/feature.

We should not underestimate the difficulties in achieving the above, each poses a difficult research challenge and how best to achieve them is an open question.

While software engineering approaches such as filtering, will continue to address feature interaction as it has in the past (albeit on a larger and more diverse scale), it will need to expand to ensure non-functional aspects such as QoS issues are incorporated into the development process *to handle feature interactions*. While it is also highly likely that software engineering can help capture the minimum information required by services or indeed ensure interfaces are used effectively, it seems unlikely that it will address the *study* of these aspects directly. In other words, the techniques employed to detect interactions are often separate from the network technologies, and so the scope for software engineering to help are limited.

Formal methods too will continue to be important. We note, however, that richer signalling and communication will, in general, make modelling less tractable (e.g. because of richer data structures). However, due to the changed nature of some standards (English text rather than formal notations) formal methods may play an increasing role in making the specified behavior explicit.

Increasingly, service providers will bundle their products. That is customers will buy a group of services from a particular provider. Using off-line techniques, service providers can check the compatibility of services belonging to the same group. This is the intra-portfolio case introduced in the previous section. Of course, service providers *may* want to perform compatibility checks on each others products. Again, off-line approaches can help, but we suspect that inter-portfolio checks are unlikely, unless the regulator steps in. However more importantly from a research perspective, formal-methods may well prove crucial in meeting the new research priorities of determining the minimum information required, the definition of interfaces, the QoS concerns, and billing.

We predict that on-line approaches will become more important as increasingly services will meet for the first time on the network. Yet this approach has not been particularly successful to date. We believe that this will change with the possibilities for direct communication and negotiation between services. With the possibility of agreed interfaces, the distributed nature of the negotiation approaches matches future telecommunications architectures closely. Thus we expect a great potential especially for negotiating approaches. This development is helped by protocols such as SIP which carry information about the call in their messages. Clearly much more work is required, for example, to limit the need for a fixed negotiation path. Expertise from the intelligent agent domain is likely to have a heavy impact on this work.

Perhaps the greatest gains will be had if these approaches can work jointly in a hybrid approach. For example, formal methods could be applied to an (off-line) analysis of service interfaces to determine *guidelines*. For example, as a result of performing the analysis, a guideline might have the form – a service offering a message "x" or action "y" is likely to interact with a service offering a message "u" or action "v". At present, this approach is simply not possible, without agreed interfaces between services.

In summary, we believe that software engineering approaches will continue to be used, however, it is unlikely that the emerging networks will alter the current research directions. The application of formal methods approaches appears to be limited to the intra-portfolio case. However, formal methods could help addressing the issues of determining the minimum information required to be exchanged between services. Due to the changing business model, on-line techniques are expected to rapidly gain importance. Research efforts into the area of feature interaction should reflect this. Furthermore, we feel that hybrid approaches, operating an on-line mechanism aided with guidance from either formal methods or service engineering approaches could prove profitable, although this view must be tempered with the limited work on hybrid approaches to date. However, these views are framed by our earlier question: *Who, if anyone, is held responsible for the correct interworking of services?*

This question, and its answer(s) is fundamental to future technical work. Yet we must be clear that the answer will be determined largely by social, commercial and possible legal forces.

10 Summary

We have presented the state of the art of the field of feature interactions in telecommunications services, concentrating on three major research trends: software engineering approaches, formal methods, and on-line techniques. The contributions of each are discussed, although there is some difficulty in assessing the industrial impact. Much of the work to date concentrates on POTS.

The telecommunications world is changing rapidly, brought about by market deregulation and technological advances. We have outlined some of these changes and the impact of the new, emerging architectures on the feature interaction problem was considered. In the emerging architectures, services will have the ability to communicate with each other and performance and billing concerns will gain prominence.

A forecast is made about how research in feature interactions needs to readjust to address the new challenges posed by the emerging architectures. From a technical perspective, the main conclusions are that the exchange of information between services must be achieved through agreed interfaces. This in turn will allow for a formal analysis of those interfaces which will inform guidelines for use, and the development of a wide variety of on-line detection, resolution and negotiation techniques. However, crucially, business relationships and responsibilities for the correct service interworking still need to be defined.

References

[1] 3GPP. The 3rd Generation Partnership Project `http://www.ieee-pin.org`.

[2] R. Accorsi, C. Areces, W. Bouma, and M. de Rijke. Features as constraints. In *[22]*, pages 210–225, May 2000.

[3] I. Aggoun and P. Combes. Observers in the SCE and SEE to detect and resolve feature interactions. In *[36]*, pages 198–212, June 1997.

[4] M. Amer, A. Karmouch, T. Gray, and S. Mankovskii. Feature interaction resolution using fuzzy policies. In *[22]*, pages 94–112, May 2000.

[5] D. Amyot, L. Charfi, N. Corse, T. Gray, L. Logrippo, J. Sincennes, B. Stepien, and T. Ware. Feature description and feature interaction analysis with use case maps and lotos. In *[22]*, pages 274–289, May 2000.

[6] M. Arango, L. Bahler, P. Bates, M. Cochinwala, D. Cohrs, R. Fish, G. Gopal, N. Griffeth, G. E. Herman, T. Hickey, K. C. Lee, W. E. Leland, C. Lowery, V. Mak, J. Patterson, L. Ruston, M. Segal, R. C. Sekar, M. P. Vecchi, A. Weinrib, and S. Y. Wuu. The Touring Machine System. *Communications of the ACM*, 36(1):68–77, January 1993.

[7] P. K. Au and J. M. Atlee. Evaluation of a sate-based model of feature interactions. In *[36]*, pages 153–167, June 1997.

[8] J. Bergstra and W. Bouma. Models for feature descriptions and interactions. In *[36]*, pages 31–45, June 1997.

[9] J. Biswas, A.A. Lazar, J.-F. Huard, K. Lim, S. Mahjoub, L.-F. Pau, M. Suzuki, S. Torstensson, W. Wang, and S. Weinstein. The IEEE P1520 Standards Initiative for Programmable Network Interfaces. *IEEE Communications Magazine*, 36(10):64 – 70, 1998.

[10] L. Blair, T. Jones, and S. Reiff-Marganiec. A Feature Manager Approach to the Analysis of Component-Interactions. In *Proceedings of FMOODS, 2002*, 2002.

[11] L. Blair and J. Pang. Feature interaction – life beyond traditional telephony. In *[22]*, pages 83–93, May 2000.

[12] J. Blom. Formalisation of requirements with emphasis on feature interaction detection. In *[36]*, pages 61–77, June 1997.

[13] J. Blom, B. Jonsson, and L. Kempe. Using temporal logic for modular specification of telephone services. In L. G. Bouma and H. Velthuijsen, editors, *[15]*, pages 197–216, May 1994.

[14] M. Boström and M. Engstedt. Feature interaction detection and resolution in the Delphi framework. In *[31]*, pages 157–172, October 1995.

[15] L. G. Bouma and H. Velthuijsen, editors. *Feature Interactions in Telecommunications Systems*. IOS Press (Amsterdam), May 1994.

[16] K. H. Braithwaite and J. M. Atlee. Towards automated detection of feature interactions. In *[15]*, pages 36–59, May 1994.

[17] J. Bredereke. Families of formal requirements in telephone switching. In *[22]*, pages 257–273, May 2000.

[18] F. Bruhns. Parlay – the API for secure and open access to networking functionality for third party applications outside the network. *Proceedings of the 6th International Conference on Intelligence in Networks (ICIN 2000)*, January 2000.

[19] G. Bruns, P. Mataga, and I. Sutherland. Features as service transformers. In *[77]*, pages 85–97, September 1998.

[20] R. J. A. Buhr, D. Amyot, M. Elammari, D. Quesnel, T. Gray, and S. Mankovski. Feature-interaction visualization and resolution in an agent environment. In *[77]*, pages 135–149, September 1998.

[21] M. Cain. Managing run-time interactions between call processing features. In *IEEE Communications Magazine*, pages 44–50, February 1992.

[22] M. Calder and E. Magill, editors. *Feature Interactions in Telecommunications and Software Systems VI*. IOS Press (Amsterdam), May 2000.

[23] M. Calder, E. Magill, S. Reiff-Marganiec, and V. Thayananthan. Theory and practice of enhancing a legacy software system. In Peter Henderson, editor, *Systems Engineering Business Process Change 2*. Springer Verlag, London, 2001.

[24] M. Calder and A. Miller. Using SPIN for feature interaction analysis - a case study. *Proceedings* SPIN *2001. Lecture Notes in Computer Science*, 2057:143–162, 2001.

[25] E. J. Cameron, N. Griffeth, Y.-J. Lin, M. E. Nilson, and W. K. Schnure. A feature interaction benchmark for IN and beyond. In *[15]*, pages 1–23, May 1994.

[26] E. J. Cameron, N. Griffeth, Y.-J. Lin, M. E. Nilson, W. Shnure, and H. Velthuijsen. Towards a Feature Interaction Benchmark for IN and Beyond. *IEEE Communications Magazine*, 31(3):64–69, March 1993.

[27] E. J. Cameron and Y.-J. Lin. Feature interactions in the new world. In *[77]*, pages 3–9, September 1998.

[28] J. Cameron, K. Cheng, S. Gallagher, F. J. Lin, P. Russo, and D. Sobirk. Next generation service creation: Process, methodology, and tool integration. In *[77]*, pages 299–304, September 1998.

[29] J. Cameron, K. Cheng, F. J. Lin, H. Liu, and B. Pinheiro. A formal AIN service creation, feature interactions analysis and management environment: An industrial application. In *[36]*, pages 342–346, June 1997.

[30] C. Capellmann, P. Combes, J. Petterson, B. Renard, and J. L. Ruiz. Consistent interaction detection – a comprehensive approach integrated with service creation. In *[36]*, pages 183–197, June 1997.

[31] K. E. Cheng and T. Ohta, editors. *Feature Interactions in Telecommunications Systems III*. IOS Press (Amsterdam), October 1995.

[32] P. Combes and S. Pickin. Formalisation of a user view of network and services for feature interaction detection. In *[15]*, pages 120–135, May 1994.

[33] F. Cuervo, N. Greene, A. Rayhan, C. Huitema, B. Rosen, and J. Segers. MeGaCo: Protocol version 1.0. *Request for Comments (Proposed Standard) 3015*, 2000. Internet Engineering Task Force.

[34] L. de Bousquet, F. Ouabdesselam, J.-L. Richier, and N. Zuanon. Incremental feature validation: a synchronous point of view. In *[77]*, pages 262–275, September 1998.

[35] J. de Keijzer, D. Tait, and R. Goedman. JAIN: A new approach to services in communications networks. *IEEE Communications Magazine*, pages 94–98, January 2000.

[36] P. Dini, R. Boutaba, and L. Logrippo, editors. *Feature Interactions in Telecommunication Networks IV*. IOS Press (Amsterdam), June 1997.

[37] A. Berendt (ed.). *IP telephony - an industry at a turning point*. Ovum, May 2001.

[38] ETSI. The The European Telecommunications Standards Institute: `http://www.etsi.org`.

[39] D. Eyers. *Telecom Markets and the Recession: An imperfect Storm, Report: AV-14-9944*. Gartner Research, December 2001.

[40] A. Felty and K. Namjoshi. Feature specification and automatic conflict detection. In *[22]*, pages 179–192, May 2000.

[41] M. Frappier, A. Mili, and J. Desharnais. Detecting feature interactions in relational specifications. In *[36]*, pages 123–137, June 1997.

[42] N. Fritsche. Runtime resolution of feature interactions in architectures with seperated call and feature control. In *[31]*, pages 43–63, October 1995.

[43] M. Fry and A. Ghosh. Application level active networking. *Computer Networks*, 31:655 – 667, 1999. Elsevier Science.

[44] A. Gammelgaard and J. E. Kristensen. Interaction detection, a logical approach. In *[15]*, pages 178–196, May 1994.

[45] P. Georgatsos, T. Nauta, and H. Velthuijsen. Role of service management in service interaction handling in an IN environment. In *[36]*, pages 213–225, June 1997.

[46] A. Ghosh, M. Fry, and G. MacLarty. An infrastructure for application level active networking. *Computer Networks*, 36:5 – 20, 2001. Elsevier Science.

[47] J. P. Gibson. Feature requirements models: Understanding interactions. In *[36]*, pages 46–60, June 1997.

[48] P. Gibson. Towards a feature interaction algebra. In *[77]*, pages 217–231, September 1998.

[49] S. Gilmore and M. Ryan, editors. *Language Constructs for Describing Features.* Springer Verlag, 2001.

[50] N. Griffeth, R. Blumenthal, J.-C. Gregoire, and T. Ohta. Feature interaction detection contest. In *[77]*, pages 327–359, September 1998.

[51] N. D. Griffeth and H. Velthuijsen. The negotiating agents approach to runtime feature interaction resolution. In *[15]*, pages 217–236, May 1994.

[52] R. Hall. Feature interactions in electronic mail. In *[22]*, pages 67–82, May 2000.

[53] R. J. Hall. Feature combination and interaction detection via foreground/ background models. In *[77]*, pages 232–246, September 1998.

[54] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg. SIP: Session initiation protocol. *Request for Comments (Proposed Standard) 2543*, 1999. Internet Engineering Task Force.

[55] J. Hay and J.M. Atlee. Composing Features and Resolving Interactions. *Proceedings of ACM SIGSOFT conference 2000*, pages 110 – 119, 2000. Association of Computing Machinery.

[56] M. Heisel and J. Souquières. A heuristic approach to detect feature interactions in requirements. In *[77]*, pages 165–171, September 1998.

[57] G. Holzmann and M. Smith. Automating software feature verification. *Bell Labs Technical Journal*, 5(2):72 – 87, April 2000. Lucent Technologies.

[58] G. Holzmann and M. Smith. Software model checking: Extracting verification models from source code. *Software Testing, Verification and Reliability*, 11(2):65 – 79, 2001. John Wiley and Sons.

[59] G. J. Holzmann and M. H. Smith. A practical method for verifying event-driven software. *Proceedings of the 1999 International Conference on Software Engineering*, pages 597–605, May 1999.

[60] S. Homayoon and H. Singh. Methods of addressing the interactions of intelligent network services with embedded switch services. *IEEE Communications Magazine*, page 42ff, December 1988.

[61] J.-S. Hwang and W. A. Miller. Hybrid blackboard model for feature interactions in process planning. *Computers and Industrial Engineering*, 29(1-4):613 – 617, 1995. Elsevier Science.

[62] Y. Inoue, M. Lapierre, and C. Mossotto, editors. *The TINA book*. Prentice Hall Europe, 1999.

[63] Y. Iraqi and M. Erradi. An experiment for the processing of feature interactions within an object-oriented environment. In *[36]*, pages 298–312, June 1997.

[64] M. Jackson and P. Zave. Distributed feature composition: A virtual architecture for telecommunications services. *IEEE Transaction on Software Engineering*, 24(10):831 – 847, 1998.

[65] JAIN. `http://java.sun.com/products/jain`.

[66] J. Kamoun and L. Logrippo. Goal-oriented feature interaction detection in the intelligent network model. In *[77]*, pages 172–186, September 1998.

[67] S. Karnouskos. Agent populated active networks. In *The 2nd IEEE International Conference on Advanced Communication Technology (ICACT-2000), Muju (Korea)*, February 2000.

[68] D. O. Keck. A tool for the identification of interaction-prone call scenarios. In *[77]*, pages 276–290, September 1998.

[69] 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. IEEE.

[70] B. Kelly, M. Crowther, J. King, R. Masson, and J. DeLapeyre. Service validation and testing. In *[31]*, pages 173–184, October 1995.

[71] M. Key, S. Leask, and A. Oshisanwo. ROSA: An object-orineted architecture for open services. *BT Telecommunications Journal*, 8(4), October 1990.

[72] A. Khoumsi. Detection and resolution of interactions between services of telephone networks. In *[36]*, pages 78–92, June 1997.

[73] A. Khoumsi and R. Bevelo. A detection method developed after a thorough study of the contest held in 1998. In *[22]*, pages 226–240, May 2000.

[74] K. Kimbler. Towards a more efficient feature interaction analysis – a statistical approach. In *[31]*, pages 201–211, October 1995.

[75] K. Kimbler. Addressing the interaction problem at the enterprise level. In *[36]*, pages 13–22, June 1997.

[76] K. Kimbler. Service interaction in next generation networks: Challanges and opportunities. In *[22]*, pages 14–20, May 2000.

[77] K. Kimbler and L. G. Bouma, editors. *Feature Interactions in Telecommunications and Software Systems V*. IOS Press (Amsterdam), September 1998.

[78] K. Kimbler, E. Kuisch, and J. Muller. Feature interactions among pan-european services. In *[15]*, pages 73–85, May 1994.

[79] K. Kimbler and D. Sobirk. Use case driven analysis of feature interactions. In *[15]*, pages 167–177, May 1994.

[80] M. Kolberg and K. Kimbler. Service interaction management for distributed services in a deregulated market environment. In *[22]*, pages 23–37, May 2000.

[81] M. Kolberg, E. Magill, D. Marples, and S. Tsang. Feature interactions in services for networked appliances. In *IEEE International Conference on Communications (ICC-2002), New York, USA.*, April 2002.

[82] M. Kolberg and E. H. Magill. Service and feature interactions in TINA. In *[77]*, pages 78–84, September 1998.

[83] M. Kolberg, E. H. Magill, D. Marples, and S. Reiff. Results of the second feature interaction contest. In *[22]*, pages 311–325, May 2000.

[84] M. Kolberg, E. H. Magill, D. Marples, and S. Reiff. Second feature interaction contest. In *[22]*, pages 293–310, May 2000.

[85] M. Kolberg, R. Sinnott, and E. H. Magill. Experiences modelling and using formal object-oriented telecommunication service frameworks. *International Journal of Computer and Telecommunications Networking*, 31(23-24):2577–2592, 1999.

[86] M. Kolberg, R. O. Sinnott, and E. H. Magill. Engineering of interworking tina-based telecommunications services. *Proceedings of IEEE Telecommunications Information Networking Architecture Conference*, April 1999. IEEE Press.

[87] T. F. LaPorta, D. Lee, Y.-J. Lin, and M. Yannakakis. Protocol feature interactions. In *Proceedings of FORTE-PSTV*, 1998.

[88] A.A. Lazar. Programming Telecommunication Networks. *IEEE Network*, 11(5):8 – 18, 1997.

[89] J. Lennox and H. Schulzrinne. *Call Processing Language Framework and Requirements, RFC 2824.* Internet Engineering Task Force, 2000.

[90] J. Lennox, H. Schulzrinne, and J. Rosenberg. *Common Gateway Interface for SIP, RFC 3050.* Internet Engineering Task Force, 2001.

[91] D. Marples and E. H. Magill. The use of rollback to prevent incorrect operation of features in intelligent network based systems. In *[77]*, pages 115–134, September 1998.

[92] D. Marples, E. H. Magill, and D. G. Smith. An infrastructure for feature interaction resolution in a multiple service environment - the application of transaction processing techniques to the feature interaction problem. In *Proceedings of TINA 95 conference*, 1995.

[93] D. Marples, S. Tsang, E. H. Magill, and D. G. Smith. A platform for modelling feature interaction detection and resolution techniques. In *[31]*, pages 185–199, October 1995.

[94] M. Nakamura, Y. Kakuda, and T. Kikuno. Feature interaction detection using permutation symmetry. In *[77]*, pages 187–201, September 1998.

[95] M. Nakamura, T. Kikuno, J. Hassine, and L. Logrippo. Feature interaction filtering with use case maps at requirements stage. In *[22]*, pages 163–178, May 2000.

[96] OPENSIG. `http://www.columbia.edu/opensig`.

[97] IEEE P1520. `http://www.ieee-pin.org`.

[98] Parlay. `http://www.parlay.org`.

[99] Y. Peng, F. Khendek, P. Grogono, and G.Butler. Feature interactions detection technique based on feature assumptions. In *[77]*, pages 291–298, September 1998.

[100] D.-B. Perng and C.-F. Chang. Resolving feature interactions in 3d part editing. *Computer-Aided Design*, 29(10):687 – 699, 1997. Elsevier Science.

[101] M. Plath and M. Ryan. Plug-and-play features. In *[77]*, pages 150–164, September 1998.

[102] M. Plath and M. Ryan. Defining Features for CSP: Reflections on the Feature Interaction Contest. In *[49]*, pages 202–216, 2000.

[103] M. Plath and M. Ryan. The feature construct for SMV: Semantics. In *[22]*, pages 129–144, May 2000.

[104] C. Prehofer. An object-oriented approach to feature interaction. In *[36]*, pages 313–325, June 1997.

[105] S. Reiff. Identifying resolution choices for an online feature manager. In *[22]*, pages 113–128, May 2000.

[106] S. M. Rochefort and H. J. Hoover. An exercise in using constructive proof systems to address feature interactions in telephony. In *[36]*, pages 329–341, June 1997.

[107] Softswitch. The International Softswitch Consortium: `http://www.softswitch.org`.

[108] B. Stepien and L. Logrippo. Representing and verifying intentions in telephony features using abstract data types. In *[31]*, pages 141–155, October 1995.

[109] D.L. Tennenhouse, J.M. Smith, W.D. Sincoskie, D.J. Wetherall, and G.J. Minden. A survey of active networks research. *IEEE Communications Magazine*, 35(1):80 – 86, 1997.

[110] J. G. Thistle, R. P. Malhamé, and H.-H. Hoang. Feature interaction modelling, detection and resolution: A supervisory control approach. In *[36]*, pages 93–107, June 1997.

[111] M. Thomas. Modelling and analysing user views of telecommunications services. In *[36]*, pages 168–182, June 1997.

[112] S. Tsang and E. H. Magill. Detecting feature interactions in the intelligent network. In *[15]*, pages 236–248, May 1994.

[113] S. Tsang and E. H. Magill. Behaviour based run-time feature interaction detection and resolution approaches for intelligent networks. In *[36]*, pages 254–270, June 1997.

[114] S. Tsang, E. H. Magill, and B. Kelly. An investigation of the feature interaction problem in networked multimedia services. *Proceedings of the Third Communication Network Symposium*, pages 58–61, July 1996.

[115] S. Tsang, E. H. Magill, and B. Kelly. The feature interaction problem in networked multimedia services - present and future. *BT Technology Journal*, 15(1):235–246, January 1997.

[116] K. Turner. Formalising the chisel notation. In *[22]*, pages 241–256, May 2000.

[117] K. J. Turner. Validating architectural feature descriptions using LOTOS. In *[77]*, pages 247–261, September 1998.

[118] Sue Uglow and Ajay Gambhir. *Wholesale: New Markets For Communications Carriers And Service Providers.* Ovum, 2000.

[119] International Telecommunications Union. *ITU-T Recommendation Q.1204: Intelligent Network Distributed Functional Plane Architecture.* ITU-T, 1993.

[120] International Telecommunications Union. *ITU-T Recommendation H.323: Packed Based Multimedia Communications Systems.* ITU-T, 1998.

[121] International Telecommunications Union. H.248, 2000.

[122] G. Utas. A pattern language of feature interactions. In *[77]*, pages 98–114, September 1998.

[123] R. van der Linden. Using an architecture to help beat feature interactions. In *[15]*, pages 24–35, May 1994.

[124] VASA. The VASA Forum: `http://www.vasaforum.org`.

[125] H. Velthuijsen. Distributed artificial intelligence for runtime feature interaction resolution. *Computer*, 26(8):48–55, August 1993.

[126] H. Velthuijsen. Issues of non-monotonicity in feature interaction detection. In *[31]*, pages 31–42, October 1995.

[127] D. Vrsalovic. Intelligent, stupid, and really smart networking. *ACM netWorker*, 2(2):44 –47, 1998.

[128] D. Wetherall. Active networks vision and reality: lessons from a capsule-based system. *ACM Operating Systems Review*, 34(5):64 – 79, December 1999.

[129] T. Yoneda and T. Ohta. A formal approach for definition and detection of feature interactions. In *[77]*, pages 202–216, September 1998.

[130] P. Zave. Architectural solutions to feature-interaction problems in telecommunications. In *[77]*, pages 10–22, September 1998.

[131] P. Zave and M. Jackson. A component-based approach to telecommunication software. *IEEE Software*, 15(5):70 – 78, 1998.

[132] P. Zave and M. Jackson. New feature interactions in mobile and multimedia telecommunication services. In *[22]*, pages 51–66, May 2000.

[133] I. Zibman, C. Woolf, P. O'Reilly, L. Strickland, D. Willis, and J. Visser. Minimizing feature interactions: An architecture and processing model approach. In *[31]*, pages 65–83, October 1995.