Features and Aspects: Exploring feature-oriented and aspect-oriented programming interactions

Technical Report No: COMP-003-2003

Computing Department, Lancaster University,
Lancaster, LA1 4YR

Fatima Beltagui
beltagui@comp.lancs.ac.uk

13th May 2003

Contents

Chapter 1

Introduction

Aspect Oriented Programming is an emerging technology which provides practical support for Separation of Concerns in software development. Undesirable interactions between different concerns or aspects are an emerging research area.

Such problems may be solvable using solutions developed for dealing with feature interactions in the telephony domain. A link was demonstrated between aspect interactions and feature interactions using an aspect oriented implementation of a telephony feature model.

Work developed in the field of feature interactions may be helpful in the task of avoiding, detecting, and resolving interactions between aspects.

Chapter 2

Aspect Oriented Programming

Aspect-Oriented Software Development is concerned with providing the techniques and tools to support crosscutting concerns. As such, it takes up the ongoing challenge of dealing with the continually increasing size and complexity of software systems.

There are a variety of aspect-oriented techniques, and the field is growing steadily. However, as a research space, there are still many open problems.

2.1 Separation of Concerns

Handling Increasing Program Size and Complexity

As the complexity and size of software projects has grown, programming techniques have developed to handle the issue of scalability. Generally, modularisation is used - the system is broken up into internally coherent modules each of which offer an external narrow interface [21].

Programming language support has provided the tools for a developer to modularise his system. For example:

- *procedural languages* introduced modular structures
- *object oriented languages* further modularised code into objects which encapsulate data and the operations upon it, and offer a narrow interface
- *components* are independent modular units with well-defined interfaces, which can be composed and integrated into a generic architecture, thus encapsulating larger units which can be deployed and re-used individually [49].

These efforts have allowed progressively larger programs to be written by allowing a developer to abstract away from concrete information and operations, and deal with the details in each separately, rather than attempting to work on an entire system as one indivisible whole.

But conceptualising increasingly more complex programs is more than just an issue of breaking a program into manageable modules. The concept of *Separation of Concerns*[21] suggests that a complex problem should be divided into separate concerns, and the developer should concern herself with each one in isolation. For each separated 'concern' "this will "reduce the detailed reasoning needed to a doable amount"[21]. Separation of Concerns is thus a powerful mechanism for dealing with complex systems in a modular way.

2.2 Cross-cutting Concerns

However, Separation of Concerns and the modularisations offered by traditional programming approaches are not always compatible. Programming languages typically offer constructs for the hierarchical composition of modular units. In general, separated concerns can be mapped easily to different modules, particularly if they are functional in nature. However, some concerns are more difficult to map in this way, generally because they crosscut the modularity of other concerns. These are labelled *cross-cutting concerns*, that is, parts of a system which, although they can be identified as distinct concerns, do not conform neatly with the lines drawn by the 'generalised' modular units provided by traditional procedural, or functional or object-oriented programming languages [38].

Other important issues may not be well-localised within such generalised modules. For example, system properties involving more than one functional component such as synchronization, component interaction, persistency, security control, etc. These are usually expressed by small code fragments scattered throughout several functional components, that is *tangled* code.

Just as object-oriented programming introduced a new way of encapsulating the parts of a system in order to develop them in isolated units which provided clear interfaces, so aspect-oriented programming (AOP) provides the ability to cross-cut this modularity with another modularity, orthogonal to the object modularity, but violating it in principled ways.

Aspect-Oriented Programming

Cross-cutting aspects seem to be a natural next step in co-ordinating the complexity of ever-larger bodies of code. Aspect Oriented programming allows more than one modularity, for example, order of execution, context of use, inter-object collaboration, and so forth. Structure arises from the entangled code because the cross-cutting concerns do have modularity, it is just that their modularity cuts across the primary modularity of the system.

Aspect-Oriented programming approaches [38] present methods and techniques for decomposing problems into a number of functional components and a number of aspects that cross-cut the functional components. Then, components and aspects are composed into a system implementation. This produces code that is simpler and more maintainable, as well as increasing the flexibility, extensibility and re-usability of the separated concerns[18].

Historically, the term Aspect Oriented Programming was first used by Gregor Kiczales [36] in the early 1990's, but many different methods which developed in parallel under other names are similar enough in motivation to share the same title. Some examples of Aspect-Oriented Programming technologies include:

- *AspectJ* [52, 37]

  AspectJ is a general-purpose aspect weaver which arose from work on reflection and preliminary studies on specialized weavers. An aspect is defined as a piece of code (written in an extension of the Java programming language) which can be composed with a collection of Java code.

- *Composition Filters* [4]

  Composition Filters are based on the concept that all communication between objects is through message passing, so filters wrap around an object to change and/or re-direct the messages received and sent in order to introduce cross-cutting concerns. Composition filters have a more formal semantics defined than does AspectJ.

- *Aspectual Collaborations* [42, 39]

- *Demeter* [1]

Aspect-Oriented Software Development

Aspect-Oriented Programming has also motivated the emergence of aspect-oriented approaches for every stage in the software life cycle, including requirements engineering, specification and design. [6]

2.3 Some Concerns Regarding AOP

Aspects may seem like an ideal mechanism for handling evolution: each patch could be provided as an aspect to be added (dynamically) to an already up-and-running system, and the patches can be re-factored into a re-engineered system at the next major release. However convenient this application of aspect-oriented programming may be, it seems inadvisable, as a likely result would be code with layer upon layer of aspects correcting bug fixes in the base code and within other aspects, making it almost impossible for a developer to comprehend what the code is doing. Also, as the layers of bug fixes become more complex, the chances increase that aspects may interacting with other aspects, resulting in unexpected behaviour.

Another concern regarding aspects is that the aspect mechanism is open to being misused in order to produce spaghetti code that is entangled to a greater degree than non-aspectual code. Aspects may be used cleanly to “violate the modularity of traditional constructs in principled ways”[36], or they may be used in ways that are as cryptic and potentially “harmful” as goto statements. Aspects may be be used to enforce global variables in an otherwise clean system. Such cross-cutting behaviour, although encapsulated within specific aspects, would be inherently global, and thus seems to undermine the basic modularity of object-oriented design.

Although traditional modularisation need not be held so sacred that we abandon the concept of crossing those lines, respecting the boundaries of the functional units is advisable if the advantages of modularisation are not to be lost. Breaking the modularity should be done in principled, well-understood ways, so that aspect-oriented technologies become available for developers to use as one of a variety of tools and methods, rather than using aspect-oriented technology as a universal solution for all corrective or evolutionary tasks.

Chapter 3

Feature Interactions

Although the Feature Interaction Problem was originally an issue in the telephony domain, it is by no means confined to it[8, 12]. In the telephony domain, services that add extra functionality to a basic call are specified as features, which are optional increments of functionality to the basic telephone service. When a feature behaves differently in isolation that when it is composed with other features in a system, a *feature interaction* is said to have occurred.

The results of the Feature Interaction research community are of potential interest in the domain of aspect-oriented programming as there is a clear link between features which compose with a basic call to provide extra services and aspects which compose with base code in order to add concerns in a well-localised way.

3.1 Feature Interactions in Telephony

Telephone Systems

Since the invention of the telephone by Alexander Graham Bell in 1876, the telephone system has undergone several legislative and technological transformations [3].

Before the 1980's, telephone systems were generally operated and maintained by single national bodies [48]. In many countries, including the UK, the telephone system developed as an extension of the public postal system. In the USA, the market leader and main telephone service provider was a private company – AT&T. However, it operated almost as a wing of the American government: in return for submitting to federal regulation, AT&T was assured a position as the foremost telecommunications company in the USA.

In 1983, AT&T was dismantled by federal court action, with sections functioning as separate corporate entities. The market became more open to competition, with many separate local, regional and international service providers and vendors appearing. In the UK, the restructuring of the General Post Office in 1981 (and the later privatisation of the British Telecom section in 1984) brought its monopoly over telephone terminal equipment and value-added services to an end. In 1983, the Mercury Consortium began the operation of an independent network to compete across the full range of telecommunication services.

Similar actions in many other countries led to a proliferation of different vendors and suppliers of telephone functions and services.

In terms of technology, there were several phases of development for the telephone system:

- manual telephone switching and routing by human operators.
- electromechanical switching systems (by 1965, the (American) telephone network was completely electronic [48])
- electronic systems, using silicon chips, which were fast, durable, and cheap to maintain. Initially, service logic was hard-wired into switching systems, with centralised decisions made about generic features [50]. Call processing code ran on each telephone exchange in a network; introducing a new feature involved the expensive procedure of changing the software on each exchange.
- Stored Program Control was introduced in the mid-1960's; this programmable service logic made it easier to introduce new features and services which could control the progress of calls and connections [50]. However, the service logic was not modular, so programs were difficult to extend. Switching systems grew to contain a lot of closely-interwoven, complex code with a high degree of inter-module coupling. The functionality was orders of magnitude greater than originally intended and switching systems became increasingly difficult to maintain and enhance.
- In the 1970's the SS7 network architecture was introduced. It performed out-of-band signalling, using a separate digital channel to manage calls, routing and billing. This allowed more data to be transported at higher speeds, and at any time throughout a call rather than just at the beginning or end. In the SS7 protocol, the service logic was external to the switching system, located instead in the database-like Signal Control Points.
- In the 1980's the Intelligent Network (IN)[50] was introduced to answer the needs of the newly de-regulated telephone industry: rapid deployment of new features, the ability to offer service customisation to individual customers, vendor-independence, and open interfaces. A generic model of a base call could be updated by adding features which are implemented as discrete components called Service Logic Programs (SLPs). The features are increments which influence the behaviour of a basic call. Any updates to overall system functionality are achieved by loading the increment rather than by modifying the software implementing the basic service. It could be argued that the features implemented in an IN are a specialised case of aspects.

- the wireless telephone network provides the same features as the Intelligent Network, and adds several more, including roaming, hands-free operation, a different fee structure and data service capabilities like SMS (Short Message Service)[51].

- Voice over IP and Internet Telephony bring new services which integrate with and potentially interact with the traditional telephone system [12].

Features

As demonstrated in the Intelligent Network architecture, one way to handle the complexity of a large reactive system like the telephone network is to break up the functionality into conceptual chunks, or features, which are designed as separate entities. Each feature has its own specification, and its addition to a base implementation can be validated in isolation to other features. Thus each feature is implemented as a separate concern acting upon the basic call model.

In the telephone network, the base implementation is known as the basic service, or *POTS* (Plain Old Telephone Service), and is the basic means of setting up and breaking down phone calls. Figure 3.1 shows a specification of a basic call, as supplied by the Feature Interaction Contest [41].

Features and services are incrementally added units of functionality which provide extra services by modifying the basic telephone service, e.g. Call Forwarding which allows a network subscriber to re-route phone calls to another telephone number (see figure 3.2). Table 3.1 shows a selection of other basic features.

Feature Interactions

The creation of new features is a major growth area in the telecommunications industry, but feature interactions are a major obstacle to the rapid implementation and deployment of new features and services.

For example, one user subscribes to both Call Forwarding and Call Forwarding When Busy (which re-directs all phone calls received only when the subscriber is already involved in a telephone call). There would be no problem if the user were subscribed to only one or the other service alone. However, if both are enabled, they may both react to the same event when the caller is busy and receives another call.

To describe the problem more formally:

Let P_1 be a property.
Let F_1 and F_2 be features.
Let $POTS$ be the basic telephone service.
If P_1 is satisfied by $POTS \oplus F_1$, and P_1 is satisfied by $POTS \oplus F_2$,
then P_1 should also be satisfied by $POTS \oplus F_1 \oplus F_2$.

i.e. the properties of F_1 and F_2 should be preserved in the presence of other features. If they are not, then there is an interaction.

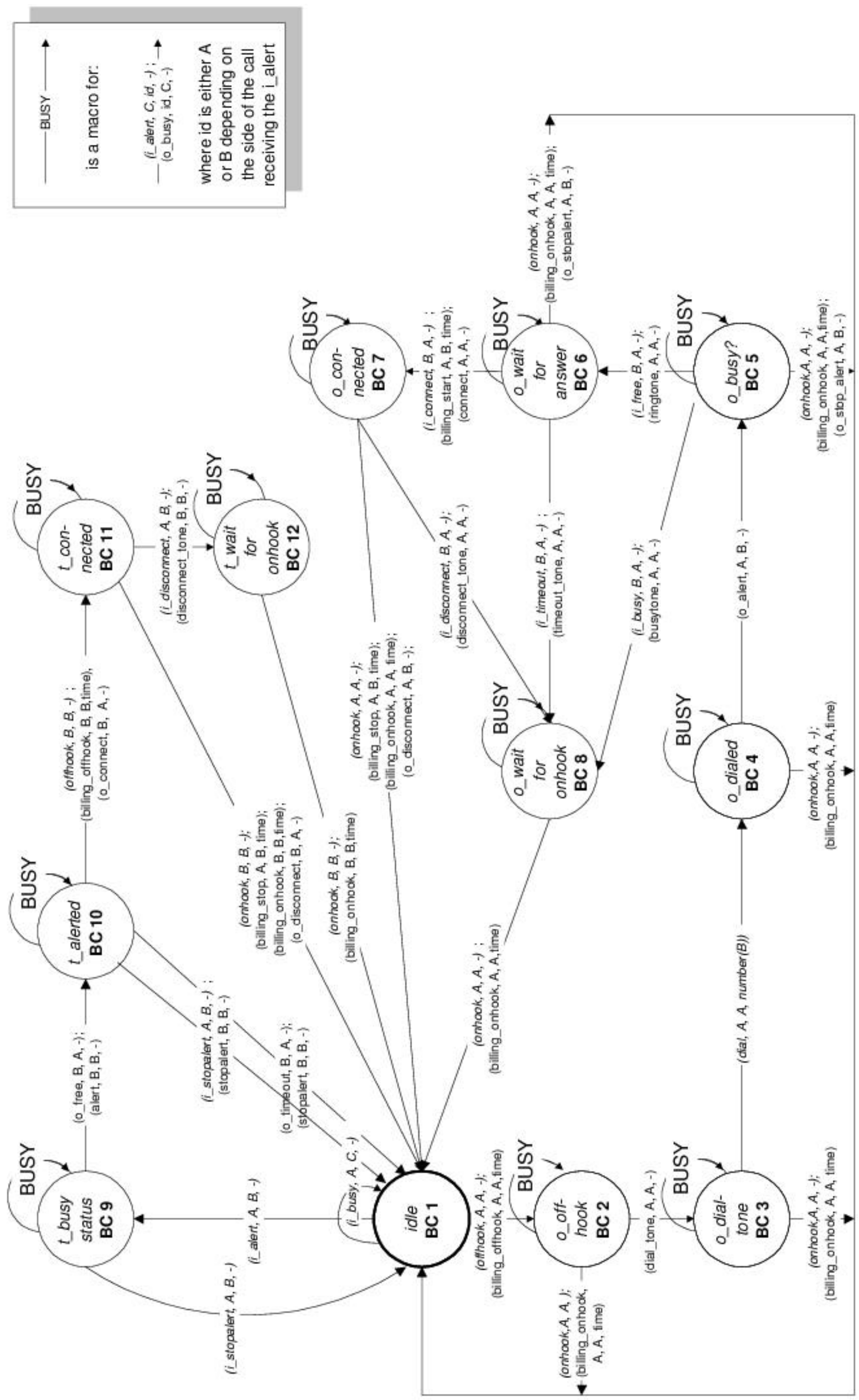

Figure 3.1: Specification of Basic Call Model in Feature Interaction Contest[41]

Feature	**Description**
Call Forwarding	Forward all calls to a particular directory number to another, specified, number.
Originating Call Screening	Restricts outgoing calls. Calls to numbers on a screening list are aborted.
Terminating Call Screening	Restricts incoming calls. Calls from numbers on a screening list are redirected.
Call Waiting	If a caller tries to call a Call Waiting subscriber whose line is busy, then the caller is placed on hold, and the subscriber is notified of the new incoming call. The subscriber can then choose to switch between the two calls.
Freephone	Redirect calls to a particular number, using all or part of the calling number and/or time of day or other state. Subscriber pays for the incoming call.
Calling Number Delivery	Delivers caller's directory number to called party.
Calling Number Delivery Blocking	Do not deliver caller's directory number to the called party. i.e. cancels Calling Number Delivery functionality.
Teen Line	Restricts outgoing calls based on the time of day. Can be overridden on a per-call basis using the proper ID code.
Three-Way Calling	Allows connection of three parties in a single conversation.
911 / 999	Only the 911 operator can terminate the call - i.e. caller cannot put operator on hold or hang up.
Automatic Callback	Subscriber automatically repeats last outgoing call when destination line line is no longer occupied.
Automatic Recall	Subscriber automatically returns last incoming call when subscribing line is no longer busy.

Table 3.1: Example Features

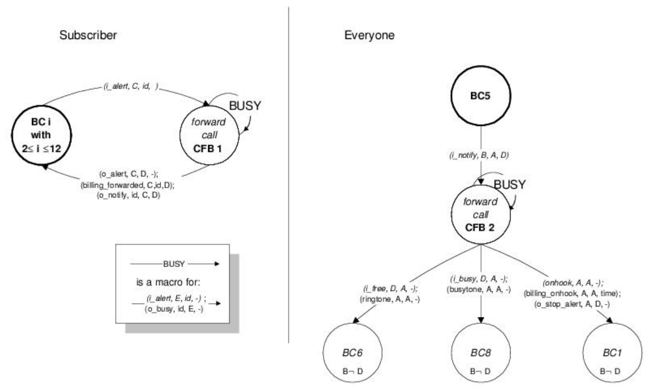

Figure 3.2: Call Forward on Busy behaviour[41]

The properties specified may be domain-specific (for example, if a user tries to call themselves, they will hear an engaged tone), or generic (for example, a system will not deadlock, or all states are reachable). Some approaches depend on a generic definition of interaction. For example, logical inconsistency or non-determinism arising from overlapping pre-conditions for processing an event may be used as indications that an interaction has occurred.

Although feature composition, $\oplus$, makes specifications easier to extend and modify, this approach de-emphasises interactions that arise in feature composition, making them implicit and harder to understand. Features implemented as modular increments, and the interfaces between them are becoming more and more explicit, but the interactions between them remain implicit.

The above definition is not enough on its own to define the Feature Interaction problem, as not all such interactions are undesirable. Some features deliberately alter or override some aspects of other features, e.g. Calling Number Delivery Blocking is designed to override Calling Number Delivery, and interactions between most features and the basic call service are desirable, as we want the system to behave differently when new features are appended.

Feature interaction is necessary and inevitable in feature-oriented specification, because so little can be accomplished by features that are completely independent. An undesirable feature interaction is one which causes specification to be incomplete, inconsistent or un-implementable, or that causes overall system behaviour to be undesirable. Non-associativity of feature composition can also be considered a feature interaction problem - for features to be truly optional, their compositions must yield the same behaviour when performed in any order. Some forms of nondeterministic behaviour can also be considered feature interaction problems (see above example of CF and CFB). A broader definition of feature interaction may include situations where resulting joint behaviours not what the user would reasonably expect from understanding each individual feature alone.

One example of a feature interaction occurs when a user subscribes to both Call Forwarding(which always re-directs telephone calls received to a specified number) and Call Forwarding When Busy (which only re-directs phone calls received when the subscriber is already involved in a telephone call). Each feature functions correctly when it is the only one enabled. However, when both are enabled, they can both react to the same event (a call attempt received when the subscriber is already engaged in a telephone call) in a non-deterministic way. This problem could be resolved, for example, by giving one feature precedence over the other.

Implicit in this example is the detection of the interaction before a resolution strategy can be used to handle it. However it is difficult to detect interactions in the telephone system, because of its large, distributed, long-lived, multi-service provider nature. It is very difficult to test all feature combinations in such a system because:

- features are added incrementally to the system by different developers,
- features are developed and tested outside the dynamic system they will be used in,
- the combinations grow exponentially in the number of features

Causes of Feature Interactions

Possible causes of specific feature interactions include [15]:

- difficult to specify service goals unambiguously in a natural language, so service goals may be incomplete, ambiguous or conflicting. Different goals may be associated with a single service, and service goals may be ambiguous.
- violation of assumptions inherent in original feature definition. Assumptions may include : how network components named and accessed, how calls are controlled, signalling protocols used, etc. As the network and its services evolve, these assumption may be violated and the services depending on out-of-date assumptions may no longer work.
- limited signalling capabilities and limitations on network components. Most telephone handsets can send only 14 signals to the switch: *, #, 0 - 9 , flash-hook, disconnect (length of time pressed may be all that distinguishes a flash-hook from a disconnect. Hence, input and output signals are assigned more than one meaning, which may lead to different effects in response to the same event.
- problems dealing with large, distributed, reactive system. e.g. resource contention, timing, race conditions.

- users may not anticipate all the implications of activating a feature or combination of features.
- user may not aware of some subtle differences in telecommunications terminology, for example, that the directory number and dialled number are not necessarily equivalent, for example, in the Freephone service the free call is forwarded to another directory number.
- services may be undocumented, poorly understood or required to interwork with services from third party legacy systems.
- number of potential feature interactions increases exponentially with the number of features in the system.

Several of these are issues that generally arise in reactive software systems, and thus may be transferable to feature interactions in non-telephony domains.

3.2 Feature Interactions Beyond Telephony

Although the term 'feature interaction' originates in the domain of telecommunications, but is by no means restricted to this domain. The phenomena of feature interaction can occur in any large and distributed system which is subject to continuous changes, or exhibits similar characteristics of real-time, event-driven operation.

The telephone network is a particularly acute example of this general problem, as it exhibits several challenging software characteristics in one domain:

- Large - international telecommunication system is world's largest distributed computing system [15].
- Real-time
- Concurrent
- Distributed Control
- Distributed Data
- Enormous state spaces
- High reliability, availability and fault tolerance
- Heterogeneous - de-regulated market encourages multiple providers.
- Long-lived and rapidly evolving

Feature Interactions have been identified in several domains other than telephony, for example email [32], the internet, multimedia systems, and mobile systems [12].

3.3 Handling Feature Interactions

Feature Interactions, particularly in the telecommunications area, have been the topic of a great deal of research effort. Literature dealing with the feature interaction problem in the telecommunications domain concentrates on three different sets of methodologies - software engineering, formal methods and online techniques [14].

Ad Hoc Methods

Interactions may be detected (and resolved) in an ad hoc manner at many different stages in the creation and deployment of a new feature [15] and be regarded as a software quality problem in different ways. A service engineer may view an interaction in terms of signal ambiguities to be resolved using functional signalling, while a requirements writer may regard the problem as a consequence of the ambiguous or incomplete specification or requirements, to be resolved using tools to aid in the creation of concise, accurate requirements. Software developers may liken the problem to extensibility issues; the needs of new services have not been anticipated by designers of the existing system. Distributed system researchers may regard the problem as one of managing issues of naming, control strategies and race conditions. Artificial Intelligence researchers may regard the problem as conflicting customer needs and/or policy objectives that can often be resolved by negotiation.

However, good techniques for modelling and analysing features develop further techniques for predicting and resolving interactions. It is useful to be able to verify that two (or more) services do not interact, or that they interact exactly as expected.

Software Techniques

Software engineering approaches mainly concentrate on importing established general techniques or introducing adapted process models to eliminate interactions. These methods tend to concentrate on designing and developing features which are free from feature interactions.

Formal Methods

Formal methods in general tend to work on detecting feature interactions. Table 3.2 shows some examples of formal methods techniques used.

Foreground/Background Models

One particular method which appears well-suited to aspect interactions is Hall's method using ISAT (Interactive Specification Acquisition Tools)[35]. A foreground/background model is used, with the Basic Call as the background model, and each feature as a separate foreground model. They are composed using a

FSM (Finite State Machines). A simple finite state machine is composed of states connected by transitions. The relative simplicity of finite state machines makes them especially suitable for formal analysis.

SDL (Specification and Description Language) is based on an extended finite state machine model, supplemented by features for specifying abstract data types. SDL has received more attention from industry than any other formal model and is well-supported by tools, some of which are commercially available.

LOTOS (Language of Temporal Ordering Specification) is a formal specification technique for specifying concurrent and distributed systems. It consists of a language for specifying processes and an algebraic specification language.

Estelle (Extended Finite State Machine Language). An Estelle specification defines a system of hierarchically-structured state machines. The state machines communicate by exchanging messages through bi-directional channels between their communication ports.

Estrel. A synchronous programming language based on a perfectly synchronous concurrency model in which concurrent processes are able to perform computations and exchange information in zero time.

Z. A (non-executable) formal specification notation based on set theory and first order predicate logic.

Temporal Logic. A formal specification language for the description and analysis of time-independent and behavioural aspects. An extension of conventional propositional logic that incorporates special operators that cater for time. There are many temporal logics, including Linear Temporal Logic (LTL) and ACTL* (an action-based temporal logic).

Promela is a protocol validation language. The SPIN model-checking tool validates Promela specifications.

Others. There are many other formal techniques, including Larch, TLA (Temporal Logic of Actions), etc., but they have received rather limited attention from the telecommunications community.

Table 3.2: Formal Techniques [19]

(textual) "foreground merge" of the features to produce a single unified foreground. Merging the models may cause inconsistencies, which are isolated as potentially undesirable interactions. Hall classifies the interactions found as follows:

- Type I interactions: inconsistent next states or output conflicts in two different models. These are automatically detected using the ISAT tool CHECK-FOR-BINARY-CONFLICTS.
- Type II interactions: state invariant or other correctness property. It holds true for one feature, but no longer holds in the merged model. No effort was made to detect these.
- Type III interactions are any remaining inconsistencies, for example, low level infinite wait states, and high level variations from desired behaviour. These were previously detected opportunistically and only if an illustrative scenario was suggested by some other tool or technique. The updated detection method was to design test scenarios for each feature pair so that each would have the behaviour of the first feature and together the set would cover all the behaviours of the second feature. The majority of these interactions were found using this method.

The same method has also been used to detect feature interactions in E-mail systems [35].

Online Methods

Online techniques generally concentrate on run-time detection and resolution of feature interactions. In one example of an online technique[46], the author points out that the feature interaction problem is in fact an intersection of three distinct problems - feature interactions in telecommunication systems, legacy systems and the multi-vendor nature of the market. Thus, the aim is to automatically detect and resolve interactions at run time, as the traditional approach of pragmatically finding and resolving features in the centralised code body is not feasible with legacy and third party code and documentation.

Chapter 4

Aspects and Interactions

(Feature) interactions also appear between composed aspects. To demonstrate the link between features and aspects, a Java implementation of a basic POTS system was extended with several features (Call Forward on Busy, Teen Line, and Ring Back when Free), and implemented as aspects using AspectJ [2]. The models used were those defined in the second Feature Interaction Contest [41].

A data structure was used to model a labelled state transition graph (see Figure 3.1) for the Basic Call, and each feature was implemented as a set of states and transitions which were added to the existing call graph. An example is the Call Forward on Busy state diagram in Figure 3.2. It was apparent that describing telephone features as aspects was simple and natural. This is not surprising as both aspects and features can be viewed as units composed with a base system.

The problematic limited signalling capabilities of telephone systems are not apparent in aspects. However, several different aspects may attempt to affect the same piece of code structure, resulting in similar interactions caused by contention over the same resources. For example, the Ring Back When Free (RBF) feature mostly consists of intercepting all routes to the IDLE state, and changing the BUSY behaviour in order to ring back the caller once the line is free. However, this feature only alters the behavious of all routes to IDLE from the base call states — so all states added by other features will not incorporate the Ring Back When Free functionality. This is an example of a very specific type of interaction: a feature (or an aspect) attempts to impact all instances of a particular action or piece of data which it is aware of, and neglects new instances added by other aspects, or during evolution, or for any other purpose.

Another interaction was discovered between Call Forward Busy (CFB) and Ring Back When Free (RBF). Given the opportunity, both attempt to react — in different ways — to the same circumstance: an incoming call when the line is currently engaged. In terms of features, the implementation will determine which feature is activated, or whether both are permitted to react. In the AspectJ implementation, whichever active feature was last weaved into the behaviour of the base call had priority in reacting to events.

Bibliography

[1] Brad Appleton. Introducing Demeter and its laws. `htpp://www.enteract.com/~bradapp/docs/demeter-intro.html`.

[2] Aspectj programming guide. `http://aspectj.org/doc/dist/progguide/index.html`, October 2001.

[3] Fatima Beltagui. Modelling telephone calls: A call process model for feature interaction detection. Undergraduate Final Year Project, April 2000.

[4] Lodewijk Bergmans and Mehmet Aksit. Composing multiple concerns using composition filters. Communications of the ACM, October 2001. Also available as: `http://trese.cs.utwente.nl/publications/papers/CF_superimposition_bergm%ans_aksit.pdf`.

[5] Gordon S. Blair, Lynne Blair, Anders Anderson, and Trevor Jones. A formal view of aspects in the development of component-based distributed systems.

[6] Gordon S. Blair, Lynne Blair, Awais Rashid, Ana Moreira, Joao Arujo, and Ruzanna Chitchyan. Engineering aspect-oriented systems.

[7] Lynne Blair and Gordon Blair. A tool suite to support aspect-oriented specification. In *ECOOP Workshops*, pages 296–297, 1999. Also available as: `trese.cs.utwente.nl/aop-ecoop99/papers/blair.pdf`.

[8] Lynne Blair, Gordon Blair, Jianxang Pang, and C. Efstratiou. Feature interactions outside a telecom domain. In *Workshop on Feature Interactions in Composed Systems at ECOOP*, 2001. Also available as: `http://www.comp.lancs.ac.uk/computing/users/lb/Postscript/FICS01.ps.gz`.

[9] Lynne Blair and Gordon S. Blair. The impact of aspect-oriented programming on formal methods (position paper). In *ECOOP Workshops*, page 436, 1998. Also available as: `http://www.comp.lancs.ac.uk/computing/users/lb/Postscript/AOP98.html`.

[10] Lynne Blair and Mattia Monga. Reasoning on AspectJ programmes. In *3rd Workshop on Aspect-Oriented Software Development*, pages 45–50, Essen, Germany, March 2003. German Informatics Society.

[11] Lynne Blair and Ken J. Turner. Case for support - SAFES (service advances for e-science), 2003.

[12] Lynny Blair and Jianxiong Pang. Feature interactions — life beyond traditional telelphony. In Calder and Magill [13], pages 83–93. Also available as: `http://www.comp.lancs.ac.uk/computing/users/lb/Postscript/FIW00.ps.gz`.

[13] M. Calder and E. Magill, editors. *Feature Interactions in Telecommunications and Software Systems VI*. IOS Press, 2000.

[14] Muffy Calder, Mario Kolberg, Evan H. Magill, and Stephan Reiff-Marganiec. Feature interaction: A critical review and considered forecast. *Computer Networks*, 41(1):115–141, January 2003. Also available as: `http://www.cs.stir.ac.uk/~srm/publications/comnet-review.pdf`.

[15] E.J. Cameron, N.D. Griffeth, Y-J Lin, M.E. Nilson, W.K. Schnue, and H Velthuijsen. A feature interaction benchmark for IN and beyond. In *Proceedings of Feature Interactions in Telecommunications and Software Systems*, pages 1–23. IOS Press, 1994.

[16] Edmund M. Clarke, Jr., Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, 1999.

[17] James C. Corbett, Matthew B. Dwyer, John Hatcliff, Shawn Laubach, Corina S. Psreanu, Robby, and Hongjun Zheng. Bandera: Extracting finite-state models from Java source code. In *International Conference on Software Engineering*, pages 439–448, June 2000. Also available as: `http://ksu.edu/~dwyer/papers/bandera.ps`.

[18] Krysztof Czarnecki and Ulrich W. Eisenecker. *Generative Programming. Methods, Tools, and Applications*, chapter 8, Aspect-Oriented Programming, pages 251–331. Addison Wesley, 2000.

[19] F. Dietrich and J.-P. Hubaux. Formal methods for communication services: meeting the industry expectations. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 38(1):99–120, 2002.

[20] Edsger W. Dijkstra. Go to statement considered harmful. *Communications of the ACM*, 11(3):147–148, March 1968. Also available as: `http://www.acm.org/classics/oct95/`.

[21] Edsger W. Dijkstra. *A Discipline of Programming*, chapter 27, In Retrospect, pages 209–217. Prentice-Hall, 1976.

[22] Rémi Douence, Pascal Fradet, and Mario Sűdholt. Detection and resolution of aspect interactions. Research Report 4435, Unité de recherche INRIA Rennes, IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex, France, April 2002. Also available as: `http://www.inria.fr/rrrt/rr-4435.html`.

[23] Remi Douence, Olivier Motelet, and Mario Sudholt. A formal definition of crosscuts.

[24] Rémi Douence, Olivier Motelet, and Mario Sűdholt. Aspect oriented programming from a monitoring perspective. Technical Report 01/4/INFO, Status of Work on AOP at the OCM group, Ecole des Mines des Nantes, April 2001. Also available as: `http://www.emn.fr/info/recherche/publications/RR01/01-4-INFO.pdf`.

[25] Robert E. Filman. What is aspect-oriented programming, revisited. In *ICSE Workshop on Advanced Separation of Concerns*, 2001. Also available as: `http://ic.arc.nasa.gov/~filman/text/oif/aop-is2.pdf`.

[26] Robert E. Filman and Daniel P. Friedman. Aspect-oriented programming is quantification and obliviousness. 2000. Also available as: `http://ic.arc.nasa.gov/\~filman/text/oif/aop-is.pdf`.

[27] Free on-line dictionary of computing. `http://foldoc.doc.ic.ac.uk/foldoc/index.html`.

[28] Pascal Fradet and Mario Sudholt. Aop: Towards a generic framework using program transformation and analysis. In *3rd AOP Workshop, ECOOP'98 Workshop Reader*, volume 1542 of *Lecture Notes in Computer Science*, pages 394–397, July 1998. Also available as: `www.irisa.fr/lande/fradet/PDFs/AOP98-long.pdf`.

[29] Pascal Fradet and Mario Sudholt. A framework for the detection and resolution of aspect interactions. In *ACM SIGPLAN/SIGSOFT Conference on Generative Programming and Component Engineering (GPCE'02)*, October 2002. Also available as: `ftp://ftp.inria.fr/INRIA/publication/publi-pdf/RR/RR-4435.pdf`.

[30] Phillip Greenwood and Lynne Blair. Aspect testing framework: Experience report combining use of AspectJ and reflection.

[31] William G. Griswold and Mehmet Aksit, editors. *Proceedings of the 2nd international conference on Aspect-oriented software development*, number 2. ACM Press, 2003.

[32] Robert J. Hall. Feature interactions in electronic mail. In Calder and Magill [13], pages 67–82. Also available as: `ftp://ftp.research.att.com/dist/hall/papers/isat/fi-in-email-fiw00.ps`.

[33] John Hatcliff and B. Matthew Dwyer. Using the Bandera tool set to model-check properties of concurrent java software. In *13th International Conference on Concurrency Theory*, Lecture Notes in Computer Science. Springer-Verlag, August 2001.

[34] John Hatcliff, Matthew B. Dwyer, and Hongjun Zheng. Slicing software for model construction. *Journal of Higher-order and Symbolic Computation*, 13(4), December 2000.

[35] Robert J.Hall. Feature combination and interaction detection via foreground/background models. *Computer Networks*, 32:449–469, 2000. Also available as: `ftp://ftp.research.att.com/dist/hall/papers/isat/feature-interactions-c%omnet00.ps`.

[36] Gregor Kiczales and Ealonor Datu. DARPA-ITO Sponsored Research 1999 Project Summary: A-O Programming, Xeroc Palo Alto Research Centre, October 2001. Also available as: `http://www.darpa.mil/ito/psum1999/J138-0.html`.

[37] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, and Mik Kersten. An overview of AspectJ. volume 2072, pages 327–355, 2001.

[38] Gregor Kiczales, John Lamping, Anurag Menhdhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In Mehmet Akşit and Satoshi Matsuoka, editors, *Proceedings European Conference on Object-Oriented Programming*, volume 1241, pages 220–242. Springer-Verlag, Berlin, Heidelberg, and New York, 1997. Also available as: `http://www2.parc.com/csl/groups/sda/publications/papers/Kiczales-ECOOP9%7/for-web.pdf`.

[39] Sergei Kojarskski, Karl Lieberherr, David H. Lorenz, , and Robert Hirschfeld. Aspectual reflection. To appear in AOSD'03 SPLAT Workshop.

[40] Mario Kolberg, Evan H. Magill, Dave Marples, and Stephan Reiff. Results of the second feature interaction contest. In Calder and Magill [13], pages 311–326. Also available as: `http://www.cs.stir.ac.uk/\~srm/publications/contest-res.ps.gz`.

[41] Mario Kolberg, Evan H. Magill, Dave Marples, and Stephan Reiff. Second feature interaction contest. In Calder and Magill [13], pages 293–310. Also available as: `http://www.cs.stir.ac.uk/~srm/publications/contest-description.ps.gz`.

[42] Karl Lieberherr, David Lorenz, and Johan Ovlinger. Aspectual collaborations – combining modules and aspects. *Journal of British Computer Society*, 2003. Also available at: `http://www.ccs.neu.edu/research/demeter/papers/ac-aspectj-hyperj`.

[43] Theo Dirk Meijler and Oscar Nierstrasz. Beyond objects: Components. In M. P. Papazoglou and G. Schlageter, editors, *Cooperative Information Systems: Current Trends and Directions*, pages 49–78. Academic Press, 1997. Also available as: `http://homepages.cwi.nl/~marcello/SAPapers/MN97.pdf`.

[44] Mattia Monga, Fatima Beltagui, and Lynne Blair. Investigating feature interactions by exploiting aspect oriented programming. 2002.

[45] Harold Ossher and Gregor Kiczales, editors. *Proceedings of the 1st International conference on Aspect-oriented software development*, number 1. ACM Press, April 2002.

[46] Stephan Reiff-Marganiec. *Runtime Resolution of Features Interactions in Evolving Telecommunications Systems*. PhD thesis, Department of Computing Science, University of Glasgow, May 2002. Also available as: `http://www.cs.stir.ac.uk/~srm/publications.php`.

[47] Marcela Shiman and Shmuel Katz. Superimpositions and aspect-oriented programming. To appear in the BCS Computer Journal. Also available as: `http://www.cs.technion.ac.il/~katz/cj.ps`.

[48] Bruce Sterling. *The Hacker Crackdown : Law & Disorder on the Electronic Frontier*, chapter Crashing the System. Viking, 1992. Also available as: `http://www.eff.org/Publications/Bruce_Sterling/Hacker_Crackdown/`.

[49] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison Wesley, 1998.

[50] Web ProForum Tutorials The International Engineering Consortium. Intelligent network IN tutorial. `http://www.iec.org/online/tutorials/in/`.

[51] Web ProForum Tutorials The International Engineering Consortium. Wireless intelligent network IN tutorial. `http://www.iec.org/online/tutorials/win/`.

[52] Cristina Videira and Gregor Kiczales. Recent developments in aspectj. In *European Conference on Object–Oriented Programming*, 1998. Also available as: `http://trese.cs.utwente.nl/aop-ecoop98/papers/Lopes.pdf`.

[53] Pamela Zave. Faq sheet on feature interaction. `http://www.research.att.com/~pamela/faq.html`, 1999.