Special issue: The future of software engineering for security and privacy

Survey Paper

Feature interaction: the security threat from within software systems

Armstrong NHLABATSI[1], Robin LANEY[2], and Bashar NUSEIBEH[3]
[1,2,3]*Department of Computing, The Open University*

ABSTRACT

Security engineering is about protecting assets from harm. The feature interaction problem occurs when the composition of features leads to undesirable system behaviours. Usually, this problem manifests itself as conflicting actions of features on a shared context. Security requirements may be violated by feature interactions creating security vulnerabilities which can potentially be exploited by attackers. In this paper, we discuss the feature interaction problem and some of its possible implications for security requirements. The paper concludes that (1) the detection of the violation of security requirements by feature interactions is not different from other types of requirements - what differs is the impact of such violation; and (2) feature interaction detection approaches can be used as a means for vulnerability analysis.

KEYWORDS

Security requirements, feature interaction detection, vulnerability analysis

1 Introduction

In order to make the complexity of modern software systems manageable their functionality is increasingly being decomposed into *features.* A feature is a set of logically-related *requirements* and their *specifications,* intended to deliver a particular behavioural effect [16], [17], [76], [88]. A feature often delivers tangible end user value [44]. A *requirement* is a statement of some system behaviour that a system is expected to exhibit [90]. The implementation of a *specification* of a feature satisfies the requirements associated with the given feature. In feature-driven development [60]. the responsibility of developing features may be allocated to different development teams.

The individually developed features are then composed, based on composition requirements, to create a feature-based application. However, the composition of features may lead to the *feature interaction problem* [17], [44], where features interfere with each other's behaviour in the composition. Such interference may lead to the violation of the requirements individual features satisfied in isolation.

Received September 14, 2007; Revised December 5, 2007; Accepted December 20, 2007.

[1)] *a.nhlabatsi@open.ac.uk*, [2)] *r.c.laney@open.ac.uk*, [3)] *b.nuseibeh@open.ac.uk*

DOI: 10.2201/NiiPi.2008.5.8

The feature interaction problem has largely been explored in terms of functional requirements. This is evidenced in the book series [18], [65] and special issue journals [2], [6], [51], [66] documenting research results on proposed approaches to addressing this problem. In this paper we review the literature on feature interaction research from a Requirements Engineering perspective, with a focus on security requirements. One of the aims of security engineering research is to identify vulnerabilities in a system and design strategies to minimize the impact of potential attacks in advance [7].

Current research aimed at addressing security has focused on modelling potential attackers. However, security requirements may be violated by feature interactions which create security holes, making a system vulnerable to attacks. This paper explores research issues related to addressing the feature interaction problem when security requirements are involved. Different ways by which a security requirement may be violated by a feature interaction, which we call *interaction types*, are reviewed. For each interaction type we review the state of the literature on offline feature interaction detection approaches and identify research challenges.

Examples of violations of security requirements where feature interaction is a primary cause can be seen in different application domains. For example in the au-

tomobile domain:

> Consider a car which has an alarm system (*security feature*) and a crash protection system with air bags (*safety feature*). The alarm system enforces security of the car occupants and their valuables. When activated it ensures that the doors and windows are locked; and monitors the state of the doors; and reports any burglary activity by activating the siren. Meanwhile, the safety feature ensures that in case of a crash, there is minimal loss of life. It achieves this by unlocking all doors in the event that an impact occurs on the front bumper.
>
> Let us consider a scenario where these features could interact. Assume the car is stationery at a traffic intersection with all doors locked by the *Security* feature. If a thief hits the front bumper with a big hammer, the *Safety* feature will unlock the doors allowing the thief to gain entry into the car.

This feature interaction may not be obvious to detect until a scenario such as the one above occurs. In this example we have assumed that safety has a higher priority than security. Without such priority a non-deterministic behaviour would result as both features try to gain control of the doors.

Such violations of safety and security requirements have common characteristics with that of privacy requirements [8]. The effects of such violations may be irreversible [74]. For example, failure of the safety feature to unlock the door because it has been blocked by the security feature may lead to the emergency personnel being unable to get to the passengers in time in case of a genuine crash.

The example highlights two problems. The first problem is how to detect, during composition, that satisfying the safety requirement will compromise the security requirement and vice versa? How do we detect during composition that having both security and safety features share control of the doors would lead to an undesirable interaction? In the event of a genuine crash it is desirable for safety to compromise security. However, this is undesirable in the case of a crash 'simulated' by a burglar. Thus, the second problem is once we know that safety compromises security; how to design the safety feature so that it is possible to differentiate between a real and faked accident. This example is not an isolated incident. Anderson [7] examines cases in which security vulnerabilities resulting from feature interactions in Telecommunications have been exploited by attackers.

The rest of the paper is structured as follows: Section 2 discusses the feature interaction problem in software systems, and possible sources of feature interactions in requirements. Section 3 identifies types of feature interactions and reviews approaches for their detection. In section 4 we discuss possible implications of the feature interaction problem on security requirements. We conclude in section 5.

2 Feature interaction

In this section we examine properties of feature interactions. We also review sources of feature interactions in requirements.

2.1 The feature interaction problem

Few features are useful in isolation. More useful functionality is achieved when features are composed with others into a feature-based application. However, the composition of features may lead to undesirable system behaviours [44]. Undesirable here means that the exhibited behaviour violates composition requirements.

Composition requirements express the intended behaviour of a feature-based system application. One way to express composition requirements is as the conjunction of the requirements individual features satisfied in isolation [89].

The feature interaction problem is one of the major challenges of feature-based software development. Research in feature interaction deals with the avoidance, detection and resolution of feature interactions. In the next section we discuss feature interaction as a problem of sharing context.

2.2 Feature interaction as a context sharing problem

When a feature interaction occurs, some requirements satisfied, in isolation, by at least one of the features involved in the interaction are violated. The relationship between two interacting features can be expressed in the formal framework for feature interaction proposed in [32]. Fig. 1 shows this framework expressed using Jackson's adequacy argument [40] and the parallel composition notation [42].

The basic principle is that if a feature specification S_1 satisfies a requirement R_1, assuming context W_1 (1), and a feature specification S_2 satisfies a requirement R_2, assuming context W_2 (2); then their parallel composition should satisfy the conjunction of R_1 and R_2 (3). Feature interaction occurs when there are shared properties between W_1 and W_2. We call such properties the shared domain (W_s). We illustrate this with security and entertainment features from a smart home [46].

$W_1, S_1 \models R_1$ (1)

$W_2, S_2 \models R_2$ (2)

$W_s, S_1 || S_2 \models R_1 \wedge R_2$ (3)

Where

$W_s = W_1 \cup W_2$

"||" is the parallel composition operator [42]

Fig. 1 Feature interaction logical properties.

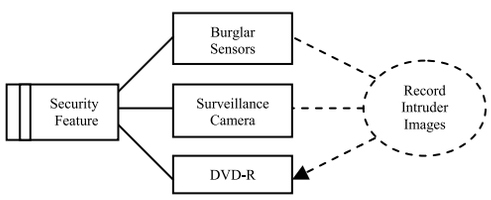


Fig. 2a Problem Diagram of Security Feature.

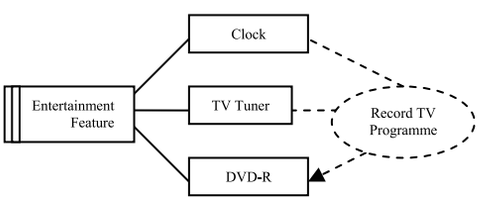


Fig. 2b Problem Diagram of Entertainment Feature.

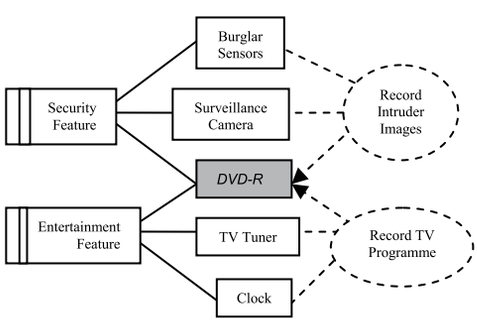


Fig. 2c Composition of Security and Entertainment Features.

Fig. 2a and 2b show problem diagrams for security and entertainment features, respectively. On detecting a burglar activity, the security feature records images of the intruders captured by the surveillance camera onto the DVD-R. The entertainment feature records a specific programme from the TV Tuner at a specified time determined by the clock.

Composition of the two features shown in Fig. 2c is expected to satisfy both the requirements of capturing burglar activities and recording TV programmes. The composition of the two features shows that they both share control of the DVD-R. Due to the nature of the DVD-R domain these two requirements may not be satisfied at the same time according to (3) in Fig. 1. Hence, there is a potential feature interaction because of the shared context.

2.3 Sources of feature interactions in requirements

Feature interaction can be characterised by their causes. Table 1 shows a summary of some causes of feature interactions that can be attributed to requirements. This is based on a taxonomy of feature interaction sources proposed by Cameron et al [21]. These include inconsistency, conflicting goals, resource contention, violation of assumptions, and overlapping preconditions.

The sources of interactions documented here only relate to the problem space.[1] In section 3.1 we discuss different types of feature interactions characterized in terms of the sources above. Exploiting the relationship between sources and types of feature interactions may make it possible to detect feature interactions by inspecting the relationship between their requirements. For example if there is an overlap between the preconditions of two features then a non-deterministic feature interaction may occur (see second row in Table 1). Thus, documentation and formalization of such relationships may provide information for early detection of feature interactions.

Indeed, the current trend in feature interaction research is to study the problem in specific domains by applying generic feature interaction detection approaches. This has led to the creation of feature interaction taxonomies of the respective domains. Examples can be seen in reports on feature interaction in smart homes [46], [57], electronic mail systems [36], SIP services [13], [24], [45], [85], web services [81]–[83], embedded systems [54], [55], policy-based systems [12], [26], [38], [67], [78], and product lines [15], [75].

Although there are similarities between the feature interactions detected in these domains, each domain presents unique challenges which have meant extending standard approaches to deal with the specific types of feature interactions. For example, Metzger [54] and Kolberg et al [46] have shown that a consideration of the environment in addressing feature interactions in embedded systems is important. This is because the environment creates dependencies between

[1] We have ignored implementation-related interactions because our interest is studying feature interactions occurring at the requirements level.

Table 1 A Summary of sources of feature interactions (from [21]).

Source	Examples
Overlapping Pre-Conditions (non-determinism)	*Call Waiting* (*CW*) with *Call Forwarding on Busy* (*CFB*) and *Voice Mail when Busy* (*VMB*): All three features are triggered by the same pre-conditions (busy subscriber) but they perform different and contradictory actions on behalf of the same user. Note: features without overlapping pre-conditions could also interact during their execution because of inconsistencies in their pre-conditions.
Requirements Inconsistency	*Security* (anti-theft system) and *Safety* (door un-locking in case of crash): Similar to above. Both *security* and *safety* shared the same sensors and are hence triggered by the same conditions, but the actions they perform as a result are inconsistent with each other.
Conflicting Goals	*Calling Line Identity Presentation* (*CLIP*) and *Calling Line Identity Restriction* (*CLIR*): CLIP delivers the calling subscribers identity, while CLIR does the opposite. This manifests itself as conflicting goals when used by two subscribers.
Violations of assumptions	*Calling Number Delivery* (*CND*) and *Unlisted Number* (*UN*): Similar to the interaction between CLIR and CLIP. CND delivers the number of the calling subscriber to the called subscriber for identification, while UN prevents an unlisted subscriber number from being delivered to a called subscriber – an example of violation of data availability assumptions.
Resource Contention	*Security* and *Entertainment* features in a smart home. The *security* feature records Images of the intruder on the VCR when a burglar is detected. The *entertainment* feature records Channel 4 news from 7:00pm to 8:00pm on the VCR. If a burglar breaks-in at 7:30pm the *security* feature will not be able to record the intruder images since the VCR is already being used by the *entertainment* feature.

features which otherwise seem independent. Specific approaches have been proposed to deal with the specific feature interactions in the environments where they occur.

Hall [37] showed that in email systems the assertion that feature interactions only occur as a violation of individual feature requirements – proposed in Fig. 1 – does not hold. He showed that feature interactions in this domain can occur without violation of individual feature requirements. For example:

> Consider the interaction between *AutoResponder* and *GroupMail* features. The *AutoResponder* feature enables automatic response to incoming email messages when the addressee is away. The *GroupMail* feature enables the creation of a virtual group of email users in a domain. For example "PostGraduateStudents" could be a group of email addresses of all post graduate students in an institution. Let the email address of this group be *postgraduatestudents@open.ac.uk*. When a message is sent to the group address, it is forwarded to each of the affiliated addresses. Now let Armstrong be a member of the above group.
>
> Consider a scenario where Armstrong is on vacation and an email is delivered to the group. The *GroupMail* feature sends this email to all addresses in the group. The *AutoResponder* feature replies by sending a response to the group email (not the originating address). When the *GroupMail* feature receives this message it forwards it to all the affiliated members. The cycle is repeated indefinitely. Note that both features have satisfied their individual requirements. However, the resulting behaviour is clearly undesirable as it ends up sending repeated messages.

Our position is that Hall's view suggests the need for an approach to the feature interaction problem that can detect non-binary interactions. In such an approach feature interaction detection would involve an analysis of the compositional effects of two features relative to a third requirement. Such analysis would be in addition to the framework proposed in Fig. 1. We discuss the implications of such an approach to security engineering in section 4.

3 Feature interaction detection

The nature of the feature interaction that occurs from the composition of features depends on the types of features involved. Feature interaction detection approaches are based on the logical relations in Fig. 1. In section 3.1 we look at feature interaction types. We discuss them in the context of security requirements and other requirements related to the smart home application domain. Approaches to detecting the feature interaction types are reviewed in section 3.2.

Table 2a Shehata et al's taxonomy.

Interaction Types	Short Description
[S1] Non-Determinism	Occurs when two features have the same pre-conditions but different post-conditions.
[S2] Dependence	The execution of one feature depends on the correct execution of another.
[S3] Override (Same pre-conditions)	Occurs when two features that have the same pre-conditions have post-conditions for one feature that change the state of the environment in such a way that the other feature does not finish executing.
[S4] Negative Impact (Same pre-conditions)	Occurs when two features with the same pre-conditions but with post-conditions that diminish each other.
[S5] Override (Linked trigger events)	Occurs between features that have linked trigger events but where the post-conditions of one feature change the state of the environment in such a way that the other feature does not finish executing.
[S6] Negative impact (Linked trigger events)	Occurs between features that have linked trigger events with post-conditions that diminish each other.
[S7] Order	The behaviour of execution of the features in one sequential composition is different from behaviour when the sequential composition is changed.
[S8] Bypass	The execution of one feature prevents the execution of another by putting the system in a state that is different from the pre-conditions of latter feature.
[S9] Infinite Looping	Occurs when features execute indefinitely by continuously triggering each other.

Table 2b Reduced taxonomy.

Interaction Types	Short Description
Non-Determinism	Same as S1
Negative Impact	Merged S4 and S6
Invocation Order	Same as S7
Bypass	Merged S3, S5 and S8
Infinite Looping	Same as S9

3.1 Requirements interaction types

One of the earliest feature interaction type taxonomies was proposed by Cameron and Velthuijsen [22]. They identified four types of feature interactions that could occur in the telephony domain. Studies on feature interactions in other domains have resulted in the refinement of this taxonomy. A notable example is the taxonomy of feature interactions in smart homes proposed by Kolberg et al [46]. Recently, these two taxonomies have been synthesized by Shehata et al [70] to a generic taxonomy which the authors claim to be applicable to most domains. This taxonomy consists of 9 feature interaction types (shown in Table 2a).

This paper uses a reduced version of Shehata et al's taxonomy. We merged S3, S5, and S8 (see Table 2a) to form a *bypass* category. S4 was merged with S6 to form a single *negative impact* category. S2 was eliminated completely.

We consider dependence to be a weaker notion of the feature interaction problem. This is because dependence implies that the dependent feature can not execute successfully in isolation. A core property of the feature interaction problem is that both features satisfy their requirements in isolation, which may be violated when they are composed. Thus, dependence is not an intrinsic property of the feature interaction problem.

The reduction resulted in 5 categories, namely: Non-determinism, Negative Impact, Execution Order, Bypass, and Looping (shown in Table 2b). We consider these categories to represent conflicts that are at the core of the feature interaction problem. This paper does not add any new categories to this taxonomy. It provides illustrations of how these types of feature interaction may impact on security requirements.

In doing so we use the concepts of pre-condition, prestate, trigger event and post-conditions as used in [70]. *Pre-conditions* describe the conditions that should be true before a feature can execute. A pre-condition consists of sets of *prestates* and *trigger events.* A *prestate* describes what the state of the system should be before a feature can execute. When a *trigger event* occurs and the *prestates* are true, a feature is executed. *Post-conditions* describe the state of the system after the execution of the given feature. In essence, a post-condition describes the effect of executing a given feature. For this reason it is stated in optative mode. [89] For example Call Forwarding on Busy (CFB) in telephony is executed when there is an *incoming call* (trigger event) while the subscriber is *engaged on another call* (prestate) and *forwards the incoming call* to a pre-specified number (post-condition).

3.1.1 Non-determinism

Non-determinism occurs when two or more feature specifications require a shared domain to engage in different behaviours simultaneously, when the domain

can engage in only one of the requested behaviours at a time [22]. By domain we mean a property of the environment that a specification of a feature uses to satisfy the requirement(s) e.g. a DVD-R in Fig. 2a. It becomes non-deterministic as to which of the required behaviours the domain should engage in. Non-determinism results from overlapping pre-conditions with inconsistent post-conditions. Overlapping pre-conditions makes it possible for features to be activated at the same time. Such overlaps may either be exactly or partially matching pre-states and trigger events.

The inconsistency of post-conditions can mean one or both of two things: a logical inconsistency between the individual feature requirements; and (or) inconsistency of the actions being performed on the domain. To illustrate this point, consider a DVD-R that is designed in such a way that it does not allow the functions of recording and playback to happen simultaneously.

Two features, F_1 and F_2, with requirements to record and playback, respectively, cannot be said to be inconsistent until we can ascertain that they are both trying to use the same DVD-R. Therefore, these features are inconsistent with respect to the domain because they are trying to engage it in behaviours that are incompatible. Incompatible here does not necessarily mean inconsistency, it simply means that the two behaviours cannot happen simultaneously and inconsistency could be one of the reasons. This problem would not occur if: each feature had their DVD-R; or if the shared DVD-R used some time-division technique which enabled it to record from more than one video source at the same time. This highlights the feature interaction problem as a "context sharing problem" as discussed in section 2.2.

3.1.2 Negative impact

Similar to a non-deterministic interaction in this type of interaction, features have overlapping pre-conditions. The difference is that in this case both features are executed but the impact of their post-conditions are inconsistent [22]. The post-conditions of one feature diminish the effects of the post-conditions of the other feature. This type of interaction can manifest as a *resource contention* [10] or *inconsistent state changes* on a shared resource (such as device in a smart house [46]).

For example, consider two features: *AirFreshMonitoring* and *ClimateControl*. The requirements for the *AirFreshMonitoring* is that when the air quality in the room is poor and it is day time the windows should be opened to refresh the room. The requirement of the *ClimateControl* feature is that during daytime the temperature in the room should be maintained at 25°C at all times by opening and closing the windows. If it is too cold or too hot outside the room (compared to inside) and the air quality in the room is poor, *AirFreshMonitoring* will open the window and this will either decrease or increase the room temperature. This has a negative impact on the requirement to maintain the room temperature at 25°C. In this example the satisfaction of the requirement is not immediate. A conflict arises when one of the features immediately close or open the windows while the requirement of the other feature is in the process of being satisfied.

3.1.3 Invocation order

An invocation order interaction occurs when the sequential composition of two or more features result into different system behaviours under different sequential compositions [70], [84]. Two features F_1 and F_2 are said to be sequentially composed if at the end of the execution of F1, the execution of F2 is started. Sequential composition can be either implicit or explicit.

With *implicit sequential composition* the sequence of feature execution results from *linked trigger* events. Two events, e_1 and e_2, are linked if the occurrence of one event leads to the occurrence of the other. For example, consider two features associated with the control of an automated door, a *DoorOpenClose* feature and a *DoorLocking* feature. The *DoorOpenClose* feature controls the opening and closing of the door. When the door is opened and a *close* event occurs, the door starts *closing* and eventually generates a *closed* event when fully closed. If an *open* event occurs while the door is closed it starts opening and generates an open event when the door is fully opened.

The *DoorLocking* feature controls the locking and unlocking of the door. It locks the door 3 seconds after the occurrence of a *closed* event and locks the door immediately when a *lock* event occurs. Similarly, this feature unlocks the door immediately when an *unlock* event occurs. *Open, Close, Lock* and *Unlock* events are generated by the user intending to enter the house where the door is mounted. This relationship between the events means that the execution of the *DoorOpenClosed* feature eventually leads to the execution of the *DoorLocking* feature. Hence the features have an *implicit sequential composition.* This is illustrated below:

> Consider a scenario in which the door is initially opened. Assume the door has close and open buttons which generate close and open events, respectively. When the user presses a close button the door is closed by the *DoorOpenClose* feature and eventually locked by the *DoorLock* feature. Define B1 to be this system behaviour.
>
> Assume a second scenario in which the

door is initially opened and the user issues a lock command which attempts to lock the door. This does not have an effect on locking the door since it is opened. If we assume that the type of lock is mechanical then we can imagine the locking bar of the mortise lock protruding after the lock event is issued. If the user presses the close button the door will start closing but will not be able to fully close because of the protruding locking bar. Define B2 to be this system behaviour.

In the former scenario both features have executed properly and satisfied their requirements. However, in the latter scenario although both features have executed, none has satisfied their requirements. In B1 the door is properly closed and locked, but in B2 the door is left unclosed! Since B1 ≠ B2, then the composition of the safety and security features exhibits an execution order interaction.

Explicit sequential composition is the type of composition in which the preceding feature in a sequential composition is designed in such a way that it directly starts the execution of the next feature. For example a security feature that automatically alerts the police through a phone call when a break-in is detected in a smart house is explicitly sequentially composed with a communications feature.

3.1.4 Bypass

One feature (F_1) *bypasses* another feature (F_2) if it changes the state of the shared environment in such a way that the new state does not match the prestates of the other feature (F_2). This prevents F_2 from executing and hence its requirements are never satisfied. To illustrate a *bypass* consider two features F_1 and F_2 with linked trigger events. Assuming F1 is triggered and starts executing. Also assume that its post-conditions are different from the prestates of F_2. This means that when the trigger event of F_2 occurs, F_2 will not be executed since the current state of the environment does not meet its prestates because of the execution of F_1.

For example consider a *Power Management* feature and a *Security* feature. The Power Management feature controls power consumption. It has parameters for monitoring the total power consumed and the rate of consumption. The total amount of power, measured in Kilowatts has a monthly limit. This feature has *adaptive power control* which ensures that power consumption does not exceed the monthly limit. Adaptive power control achieves this by monitoring and adapting power consumption by 'greedy' appliances. When an appliance consumes power at a rate higher than the average rate then that appliance is switched-off to ensure a steady consumption of power. On detecting a burglary, the Security feature raises an alarm by sounding a motorised siren. Assume a burglar is detected and the security feature starts the motorised siren which consumes power at a rate higher than the average rate. On detecting this, the power management feature switches off the power to the siren. As a result the security requirement is not satisfied and the power management feature is said to have bypassed the security feature.

3.1.5 Looping

Looping feature interactions are unique in the sense that they defy the general notion of feature interaction. In this type of feature interaction individual feature requirements are not violated. A looping interaction occurs when two features are reciprocally linked in their post-conditions and trigger events [22], [46], [70]. Two features, F_1 and F_2, are reciprocally linked if the post-conditions of F_1 create the trigger events of F_2 and vice versa. To illustrate a looping interaction, assume that F1 is triggered and starts executing and creates the trigger events of F2. Feature F2 starts executing and in turn creates trigger events for F1. This process is repeated indefinitely — creating infinite looping.

For example consider a *Cooling* feature and a *Security* feature. When the temperature inside a house is higher than that outside, the Cooling feature opens the windows and starts the fan. On detecting movements in the house the security feature raises an alarm by sounding the siren and secures windows to ensure that the burglar does not get away. Consider a scenario in which the temperature in the house is hotter than outside. This triggers the *Cooling* feature which by starting the fan creates movements in the house which are interpreted by the *Security* feature as being caused by a burglar. The security feature shuts the windows. This makes the room warm again and triggers the Cooling feature, which again starts the fan and opens the windows. This cycle continues indefinitely.

3.1.6 Relation to requirements interaction management

Robinson et al [68] proposed three properties of requirements interactions: *Basis*, *Degree* and *Direction*, and *Likelihood*. The *basis* specifies the basic elements of the feature interaction, that is, the minimum set of conditions that imply an interaction between features. This is much like the five feature interaction types we have discussed above. The *degree* specifies the impact of the interaction on the operation of the system and the *direction* specifies whether the interaction is negative (undesired) or positive (desired) with respect to the sat-

isfaction of system composition requirements. The degree and direction is a measure of the *interaction level* of a given set of features and can help in prioritizing the resolution of undesirable feature interactions. Negative interactions with a high impact should be given a higher priority than those with a lower negative impact. The *likelihood* of a feature interaction determines its probability of occurrence. These properties seem attractive criteria for evaluating feature interactions and such evaluation will vary from domain to domain and may depended on the type of feature interaction. However, we do not have systematic methods of measuring *degree* and *likelihood* of feature interactions. Current approaches can only detect that an undesirable feature interaction has occurred but can not tell us what the impact is and how likely it is that the feature interaction would occur.

The next section surveys approaches to detecting the types of feature interactions described above.

3.2 Formal approaches

Formal methods are precise languages and techniques for specifying and analyzing systems. Due to their rigour, precision, and systematic treatment they are highly desirable in the development of software systems where a high standard of safety and integrity is essential [87]. These benefits have been demonstrated for the detection of feature interactions in telecommunications features, such as in the use of Temporal Logic [28]. The application of these approaches in addressing the feature interaction problem can be classified into Logic Based, State/Model-Based, Algebraic, and Structural approaches. This classification is based on classifications proposed in Liu et al [50], Turner et al [79], and Kryvyi and Matveyeva [47].

A summary of these approaches is presented in Table 3. For each class of approach it shows the notation used for feature specification, the type of feature interaction detected, the feature interaction detection approach used, the application domain, and tool support (where available). This table shows no evidence of approaches that address the detection of looping interactions. Following is a brief summary of each class.

3.2.1 Logic

This approach involves the use of logics to describe system desired properties. Validity of properties is checked using the associated axiom system of the used logic. The commonest types logic systems used for specifying features and reasoning about their compositional behaviours are modal temporal logic and the Event Calculus. Temporal Logic expresses how the system behaviour evolves over time, making it possible to make statements about future states of the system. It can be used to reason about qualitative and quantitative temporal properties. Qualitative properties include safety properties (such as mutual exclusion and absence of deadlock) and liveness properties (such as termination and responsiveness). Examples of quantitative properties include periodicity, deadline, and delays. Calder et al [19] and Felty [28] have used temporal logic for specifying the behaviour of telecommunications features and used the model checking tool SPIN [20] to automate the process of detecting interactions.

The Event Calculus [69] is a logical language for representing and reasoning about actions and their effects. It is also being used for specifying and analysing feature-based system behaviour. An Event Calculus description relates *initiating* and *terminating* events to system states called *fluents*. A fluent is a property of the system that holds after it is initiated by an event and ceases to hold when terminated by another. An event e_1 is said to initiate a fluent f if upon occurrence of e_1, f becomes true. Meanwhile an event e_2 is said to terminate fluent f if its occurrence makes f to be false. This logic system has been used for analysing conflicts between policy specifications [9], avoiding feature interactions resulting from inconsistent smart home features [48], [49], and real-time monitoring of requirements satisfaction in service-based systems [73].

Yokogawa et al [86] propose the use of bounded model checking for the detection of feature interactions. In this approach, the problem of feature interaction is reduced to that of the propositional satisfiability decision problem [14]. Mueller [56] presents a comprehensive comparison between Event Calculus and Temporal Action Logic (TAL) [30] which could be useful as guidance in deciding which logic system to use for a given application. Giannakopoulou and Magee [31] have proposed an approach of translating event-based specifications into fluent propositions which makes them amenable to analysis with model-checking tools.

Features have also been specified as constraints on system behaviour and feature interactions defined as violation of such constraints. Accorsi [1] proposes an approach in which features are specified as constraints and model checking tools are then used to analyse the specifications for feature interactions. In [27] a constraint-based approach for performing avoidance, detection, and resolution of feature interactions is proposed. Hay and Atlee [39] propose a transitions synchronization technique called Conflict Free-Synchronisation. This technique allows features to simultaneously react to a particular situation, but disables transition combinations that conflict. Two features conflict if the combination of their transitions violates rel-

Table 3 A summary of formal approaches.

Approach	Feature Specification Notation(s)	Type of Interaction(s) Detected	Approach to FI Detection	Application Domain(s)	Tool Support	References
State/ Model-Based	Procedural Event-Based Formalism (P-EBF), Traces of Finite State Automata, MSC, and SDL	Negative Impact, and Bypass,	Constraint Satisfaction, Reachability Analysis, and Simulation	Intelligent Networks (IN) services, POTS and Distributed SIP Services.	ISAT LTSA [31]	Hall [35], Elf et al [27] Kaindl [43], Turner [77], Fu et al [29], Damas et al [25], Lorentsen et al [52], Uchitel and Chechik [80]
Logic-Based	Linear Temporal Logic Formulas and Event Calculus Descriptions	Non-Determinism and Invariant Violation (Bypass)	Model Checking, Constraints Satisfaction, and Logic Deduction.	POTS, Smart Homes, Policies	SPIN and Event Calculus Planner	Calder et al [19], Felty et al [28], Laney et al [48], Bandara et al [9], and Accorsi et al [1], Shanahan [69], Blair et al [12], [26], [64]
Algebraic	LOTOS	Bypass, Override, and Negative Impact	Constraint Satisfaction	POTS	ELUDO, CADP, and LOLA [29]	Fu et al [29], Gorse [33], and Nakamura et al [58], [59],
Structural	Petri-nets and Use Case Maps (UCMs)	Non-determinism	Simulation and reachability analysis[2]	POTS and User Interfaces for Mobile Phones	DESIGN/CPN [3]	Amyot [5], Nakamura et al [58], Jackson and Zave [41], Pomakis and Atlee [62], Kryvyi [47], Lu et al [53], and Lorentsen et al [52].

evant assertions.

3.2.2 State/model-based

State-based languages are used to model the behaviour of a feature-based system in terms of abstract machines with sets of states and transitions between the states. The machine changes from one state to another depending on the input and may produce some output in response. Some notable examples of state-based languages include the Specification and Description Language (SDL) [43], [77], and Message Sequence Charts (MSC) [25], [29], [52], [80].

SDL is an ITU z.100 standard language for analysing specifications for completeness and correctness, determining conformance of implementation to specifications, and determining consistency between specifications. It is intended for specification of complex, event-driven, real-time, and interactive applications which involve concurrent processes that communicate using discrete signals.

MSCs model system behaviour using scenario-based specifications and they focus on messages exchanged between features. The basic principle of feature interaction detection with stated-based approaches is determining state reachability [62], [71], i.e. whether all the states reachable in isolation are reachable in composition. A given state is associated with the satisfaction of certain properties; hence if the given state is not reached the satisfaction of these properties is violated.

In the approach proposed by Hall [35], a language similar to LISP is used to specify foreground and background models. A foreground model contains the essential changes introduced by features while a background models represents the static part of the system.

3.2.3 Algebraic

Algebraic approaches are similar to stated-based approaches. The only difference is that with algebraic approaches the consideration of state information is implicit and the focus is on actions that cause transitions between states. A feature-based system is modelled as a set of communicating processes with each process modelling a single feature. Each feature process describes the order in which events can occur (sequen-

[2] These approaches do not actually detect feature interaction but through their architecture they impose feature prioritization; hence ensuring that higher priority features always takes precedence in case of a conflict. This increases the predictability of system behaviour.

tially or concurrently).

An example of an Algebraic language is Language Of Temporal Ordering of Specification (LOTOS). In [29], LOTOS is used for describing feature specifications and these specifications are then translated into state transition model that describe properties that should hold either globally or locally. These properties describe required feature behaviour and their violations are considered as feature interactions. State transitions that do not lead to property violations are encoded as Message Sequence Charts.

In [33], a two-stage approach to detecting feature interactions in LOTOS specifications is proposed. The first stage is *filtering* in which possible interactions are detected by considering feature prestates, trigger events, post-conditions, and constraints. Nakamura et al [58], [59], proposes heuristics for filtering based on feature specified with Use Case Maps. The second stage is *testing*. At this stage suspect interactions identified in the filtering stage are further analysed to ascertain if they can actually occur. This approach does not guarantee that all possible interactions have been detected since the set of filtering heuristics have not been demonstrated to be complete. Moreover, it is generally accepted that testing does not guarantee the absence of feature interactions [32].

3.2.4 Structural

With structural approaches the organisation of the system is defined in terms of its components – the features. Structural approaches are useful as visual notation for representing sequences of actions and the causality among them, e.g. Use Case Maps (UCM) [5]. Structural approaches are not formal in themselves and consequently they are often accompanied by a formal underpinning which describes rules of valid connections between components. This is demonstrated in [58] where a formal link is provided from UCM to LOTOS.

Architecture-centric methods to handling feature interactions such as the DFC [41] and the Feature Stack Architecture [62] demonstrate the use of structural approaches. Both of these methods resolve nondeterministic feature behaviour by prioritizing features, ensuring that they always execute in a deterministic way. Practical application of the DFC (initially developed for Plain Old Telephone System (POTS)) has been demonstrated through its implementation in an IP telephony platform called BoxOS [13].

Petri Nets provide a graphical representation with formal semantics of system behaviour and they can deal with concurrency, non-determinism, and casual connections between events. In the approach proposed in [53], feature functionality is represented as a temporal formula and the behaviour of the featured-based system is represented as the set of all firing sequences. Feature interactions are detected by inspecting whether or not the temporal formula is violated when executing some of the firing sequences.

The CHISEL notation [77] is an informal graphical notation describing telecommunications features and services. Its graphical descriptions are supported by LOTOS and SDL.

3.3 Research issues

Formal specifications of features help improve clarity and precision [19]. Formal analysis of feature compositions allows for rigour in the detection of feature interactions [17]. Although the application of formal approaches has proven valuable in understanding feature interactions, especially in the telecommunications domain, the main challenges for applicability of formal approaches concern end-user programming. The main goal of end-user programming is to equip end-users (rather than developers) with tools for designing and composing their features. This is different from current practice in which features are designed and composed by experienced developers.

Such a development paradigm raises two issues: (1) how can formal approaches be used to capture and formalize user intentions, and (2) how to handle feature interactions between end-user defined features. A majority of feature interactions can be traced to the way user intentions are interpreted. For example resolving interaction between the Safety feature and the Security feature in the automobile example given earlier requires knowledge of what the car owner prefers. Resolution of such interactions is currently through static priorities determined in advance. However, run-time compositions may need dynamic resolutions.

Composition of features from different users will require user intentions to be captured and formalized and such information may aid resolution. The Composition Controller [48], [49] is a step in this direction as it can store alternative resolutions associated with certain composition requirements. It allows the desired resolution to be chosen at run-time. However, the alternative resolutions themselves are based on static composition requirements.

With internet telephony end users are able to create their own features. For example it is now possible for a user to define a policy which handles incoming calls depending on who is calling and when [12]. Currently, feature interactions can be managed because the development of features and their composition is under a few major firms. When definition of features is distributed among end users more sophisticated formal methods may be required for defining feature compositional be-

haviour.

4 Implications for security requirements

We now look at possible implications of the preceding discussion for security requirements. We first look at some of the distinct characteristics of security requirements.

4.1 Some characteristics of security requirements

In order to protect assets from harm security requirements constrain functional requirements [34], [38], [72]. For example, transferring financial information between accounts held in different banks is a functional requirement. To achieve integrity and confidentiality it is necessary to ensure, respectively, that: (1) the amount of money to be transferred and the account where it is to be deposited is not changed during transmission; and (2) unauthorised third parties should not know how much money has been transmitted and who is the beneficiary. Both of these security concerns constrains the way information is transmitted.

To realise a security requirement it is necessary to refine it into a functional requirement [34]. For example items in a car can be stolen by opening the doors. Thus, the requirement of preventing theft can be refined to: "keep the doors locked". This is a functional requirement whose implementation is sufficient to satisfy the original security requirement.

A security requirement may be violated by the compositional behaviour of functional requirements [17]. An example of this is a three-way feature interaction in telecommunications [44]:

> Assume a telephone user A is subscribed to *Calling Line Identity Restriction (CLIR)* which prevents his number from being displayed when calling another subscriber (a *confidentiality* feature). Also consider another user B who has two features: *Automatic Recall (ARC)* and *Itemized Billing (ITM)* active. ARC enables B to return a call to the last caller without knowing their (caller's) number. ITM is a call accounting feature which enables B to produce a list of all telephone numbers dialled over a given period. If A calls B and later B uses his *ARC* feature, the system will connect him to A and then A's number will now be listed in B's itemized bill. As a result A's CLIR feature is compromised! This is illustrated in Fig. 3. Note that the requirements of the ARC and ITM features are satisfied. However, the confidentiality requirement of the CLIR is violated.

Fig. 3 Illustration of a 3-way feature interaction.

Examples like this illustrate that the feature interaction problem may lead to violation of security requirements. Vulnerability analysis needs to address this problem. In the next section we briefly discuss some techniques for vulnerability analysis.

4.2 Vulnerability analysis

A vulnerability is a weakness in a system that can be exploited by attackers to compromise its (system's) security [7]. Vulnerability analysis is the process of identifying, quantifying, and prioritizing vulnerabilities [11], [61], [63]. A number of approaches have been proposed for vulnerability analysis. These approaches are generally classified as model-based [63], graph-based [4], [61] and constraint-based [11].

Model-based approaches construct high-level models of system components; formalize desired security-relevant properties of the composite system; and analyse system models to check for deviation from desired security properties [63]. Graph-based approaches organise attack exploits into trees or graphs [4]. Each node represents relevant system attributes such as specific vulnerabilities on various hosts in a network. Each transition in the graph represents a specific exploit that an attacker can carry-out. Finally, constraint-based approaches use constraint satisfaction to detect and eliminate cascading network paths that compromise security [11].

The main limitation of the above approaches is that they do not explicitly consider compositional effects of system components. Yet such compositional effects may lead to interactions that compromise security requirements.

5 Conclusions and future work

Security requirements have distinct characteristics. However, their refinement into functional requirements blurs this distinction with respect to the feature interaction problem. Hence the detection of feature interactions involving security requirements can be achieved with current approaches to feature interaction detection.

We have looked at whether specific characteristics of security requirements may affect the feature interaction problem. It may also be worth investigating the effect

the feature interaction problem has on the problem of security engineering? Does it make security more challenging to address? Does the composition of two features which satisfy some security requirements in isolation result in an overall system that is not secure? To address these issues research on software composition, including aspect orientation [23], hold some promise.

Acknowledgements

We are grateful to the EPSRC for providing the financial support that has made this work possible. We are grateful to Michael Jackson for his insightful review that shaped the message of this paper. We also thank Mohammed Salifu, Thun Thein, Yudistira Asnar, and Jan Jurjens for the helpful discussions that enriched the ideas in this paper. Finally, we are very thankful to the anonymous reviewers for their useful criticism that helped improve the paper.

References

[1] R. Accorsi, C. Areces, W. Bouma, and M.d. Rijke, *Features as Constraints*, in *Feature Interactions in Telecommunications and Software Systems*, M. Calder and E. Magill, Editors. IOS Press: Amsterdam. pp. 210–225 , 2000.

[2] I.F. Akyildiz, H. Rudin, L.G. Bouma, N. Griffeth, and K. Kimbler, "Special issue on the feature interactions in telecommunications systems."*Computer Networks*, vol. 32, no. 4, 2000.

[3] K. Albert, K. Jensen, and R. Shapiro, "A Tool Package Supporting the Use of Colored Nets." *Petri Net Newsletter*, vol. 32, pp. 22–35, 1989.

[4] P. Ammann, D. Wijesekera, and S. Kaushik, "Scalable, graph-based network vulnerability analysis." *Proceedings of the 9th ACM conference on Computer and communications security*, pp. 217–224, 2002.

[5] D. Amyot, "Use Case Maps as a Feature Description Notation," in *Language Constructs for Describing Features*, S. Gilmore and M. Ryan, Editors. 2001, Springer, Berlin.

[6] D. Amyot and L. Logrippo, "Special issue: Directions in feature interaction research." *Computer Networks*, vol. 45, no. 5, 2004.

[7] R. Anderson, *Security Engineering: A Guide to Building Dependable Distributed Systems*. Canada: John Wiley & Sons, Inc 2001.

[8] A.I. Antón, and J.B. Earp, "A requirements taxonomy for reducing Web site privacy vulnerabilities."*Journal of Requirements Engineering*, vol. 9, no. 3, 2004.

[9] A.K. Bandara, E.C. Lupu, and A. Russo. "Using event calculus to formalise policy specification and analysis." in *Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*. 2003.

[10] J. Bisbal and B.H.C. Cheng, "Resource-based Approach to Feature Interaction in Adaptive Software." *Proceedings of the 1st ACM SIGSOFT workshop on Self-managed systems*, pp. 23–27, 2004.

[11] S. Bistarelli, "A soft constraint-based approach to the cascade vulnerability problem." *Journal of Computer Security*, vol. 13, no. 5, pp. 699–720, 2005.

[12] L. Blair and K.J. Turner, "Handling Policy Conflicts in Call Control." in *Proc. International Conference on Feature Interaction VIII*. Amsterdam, IOS Press. 2005.

[13] G.W. Bond, E. Cheung, K.H. Purdy, P. Zave, and C. Ramming, "An Open Architecture for Next-Generation Telecommunication Services." *ACM Transactions on Internet Technology (TOIT)*, vol. 4, no. 1, pp. 83–123, 2004.

[14] L. Bordeaux, Y. Hamadi, and L. Zhang, "Propositional Satisfiability and Constraint Programming: A comparative survey." *ACM Computing Surveys*, vol. 38, no. 4, p. 12, 2006.

[15] J. Bredereke, "Configuring Members of a Family of Requirements Using Features." in *Feature Interactions in Telecommunications and Software Systems VIII*. Leister, U.K., IOS Press. 2005.

[16] J. Bredereke, "On Feature Orientation and on Requirements Encapsulation Using Families of Requirements," in *Objects, Agents, and Features*, J.-J.C.M. Mark D. Ryan, Hans-Dieter Ehrich, Ed. Springer-Verlag, Berlin Heidelberg. pp. 26–44, 2004.

[17] M. Calder, M. Kolberg, E. Magill, and S. Reiff-Marganiec, "Feature interaction: A critical review and considered forecast." *Comput. Networks*, vol. 41, no. 1, pp. 115–141, 2003.

[18] M. Calder, and E. Magill, *Feature Interactions in Telecommunications and Software Systems VI*. Amsterdam, The Netherlands, IOS Press. 2000.

[19] M. Calder and A. Miller, "Feature interaction detection by pairwise analysis of LTL properties: a case study." *Formal Methods in System Design*, vol. 28, no. 3, pp. 213–261, 2006.

[20] M. Calder, and A. Miller, "Using SPIN for Feature Interaction Analysis - A Case Study." in *Proceedings of the 8th international SPIN workshop on Model checking of software*. Toronto, Ontario, Canada, Springer-Verlag New York, Inc. 2001.

[21] E.J. Cameron, N. Griffeth, Y.-J. Lin, M.E. Nilson, W.K. Schnure, and H. Velthuijsen, "A feature-interaction benchmark for IN and beyond." *IEEE Communications Magazine*, vol. 31, no 3, pp. 64-69, 1993.

[22] E.J. Cameron and H. Velthuijsen, "Feature interactions in telecommunications systems." *IEEE Communications Magazine*, vol. 31, no. 8, pp. 18–23, 1993.

[23] A. Charfi and M. Mezini, "Using aspects for security engineering of Web service compositions." in *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on*. 2005.

[24] C. Chi and R. Hao, "Test generation for interaction detection in feature-rich communication systems." *Journal of Computer Networks: Special Issue on Feature Interaction*, vol. 51, no. 2, pp. 426–438, 2007.

[25] C. Damas, B. Lambeau, P. Dupont, and A. van Lamsweerde, "Generating annotated behavior models from end-user scenarios." *IEEE Trans. Softw. Eng.*, vol. 31, no. 12, pp. 1056–1073, 2005.

[26] P. Dini, A. Clemm, T. Gray, F.J. Lin, L. Logrippo, and S. Reiff-Marganiec, "Policy-enabled mechanisms for feature interactions: reality, expectations, challenges." *Comput. Networks*, vol. 45, no. 5, pp. 585–603, 2004.

[27] C.D. Elfe, E.C. Freuder, and D. Lesaint, "Dynamic constraint satisfaction for feature interaction." *BT Technology Journal*, vol. 16, no 3, 1998.

[28] A.P. Felty and K.S. Namjoshi, "Feature specification and automated conflict detection." *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 12, no. 1, pp. 3–27, 2003.

[29] Q. Fu, P. Harnois, L. Logrippo, and J. Sincennes, "Feature interaction detection: a LOTOS-based approach." *Comput. Networks*, vol. 32, no. 4, pp. 433–448, 2000.

[30] M. Gelfond and V. Lifschitz, "Representing action and change by logic programs." *The Journal of Logic Programming*, vol. 17, no. 2–4, pp. 301–321, 1993.

[31] D. Giannakopoulou and J. Magee, "Fluent model checking for event-based systems," in *Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering*. ACM Press, Helsinki, Finland. pp. 257–266, 2003.

[32] J.C. Godskesen, "A Formal Framework for Feature Interaction with Emphasis on Testing," in *Feature Interactions in Telecommunications Systems III*, K.E. Cheng and T. Ohta, Editors. IOS Press. pp. 21–30, 1995.

[33] N. Gorse, L. Logrippo, and J. Sincennes, "Formal Detection of Feature Interactions with Logic Programming and LOTOS." *Jornal of Software and Systems Modeling*, vol. 5, no. 2, p. 135, 2006.

[34] C.B. Haley, J.D. Moffett, R. Laney, and B. Nuseibeh, "A framework for security requirements engineering," in *Proceedings of the 2006 international workshop on Software engineering for secure systems*. ACM, Shanghai, China. pp. 35–42, 2006.

[35] R.J. Hall, "Feature combination and interaction detection via foreground/background models." *Comput. Networks*, vol. 32, no. 4, pp. 449–469, 2000.

[36] R.J. Hall, *Feature Interaction in Electronic Mail*, in *Feature Interactions in Telecommunications and Software Systems VI*, M. Calder and E.H. Magill, Editors. IOS Press, Glasgow, Scotland, UK, 2000.

[37] R.J. Hall, "Fundamental Nonmodularity in Electronic Mail." *Autom. Softw. Eng.*, vol. 12, vol. 1, pp. 41–79, 2005.

[38] H. Hamed and E. Al-Shaer, "Taxonomy of conflicts in network security policies." *Communications Magazine, IEEE*, vol. 44, no. 3, pp. 134–141, 2006.

[39] J.D. Hay and A.J. M., "Composing Features and Resolving Interactions." *ACM SIGSOFT Software Engineering Notes*, vol. 25, Issue 6, pp. 110–119, 2000.

[40] M. Jackson, *Problem frames : analysing and structuring software development problems*. ACM Press. 2001, Harlow, Addison-Wesley, 2001.

[41] M. Jackson and P. Zave, "Distributed Feature Composition: A Virtual Architecture for Telecommunications Services." *Software Engineering, IEEE Transactions on*, vol. 24, no. 10, pp. 831–847, 1998.

[42] N. Jianwei, J.M. Atlee, and N.A. Day, "Template semantics for model-based notations." *IEEE Transactions on Software Engineering*, vol. 29, no. 10, pp. 866–882, 2003.

[43] H. Kaindl, "A scenario-based approach for requirements engineering: Experience in a telecommunication software development project." *Systems Engineering*, vol. 8, no. 3, pp. 197–210, 2005.

[44] D.O. Keck and P.J. Kuehn, "The Feature and Service Interaction Problem in Telecommunications Systems: A Survey." *IEEE Trans. on Softw. Eng.*, vol. 24, no. 10, pp. 779–796, 1998.

[45] M. Kolberg and E.H. Magill, "Managing feature interactions between distributed SIP call control services." *Journal of Computer Networks: Special Issue on Feature Interaction*, vol. 51, no. 2, pp. 536–557, 2007.

[46] M. Kolberg, E.H. Magill, and M. Wilson, "Compatibility Issues between Services Supporting Networked Appliances." *IEEE Commun. Mag.*, vol. 41, no. 11, pp. 136–147, 2003.

[47] S.L. Kryvyi and L.Y. Matveyeva, "Formal Methods of Analysis of System Properties." *Journal of Cybernetics and Systems Analysis*, vol.39, no. 2, pp. 174 –191, 2003.

[48] R., Laney, M. Jackson, and B. Nuseibeh, *Composing Problems: Deriving specifications from inconsistent requirements*. The Open University: Milton Keynes, U.K., 2005.

[49] R. Laney, T.T. Tun, M. Jackson, and B. Nuseibeh, *Composing Features by Managing Inconsistent Requirements*. in *9th International Conference on Feature Interactions in Software and Communication Systems*. Grenoble, France, 2007.

[50] X. Liu, H. Yang, and H. Zedan, "Formal methods for the re-engineering of computing systems: a comparison." in *Computer Software and Applications Conference, 1997. COMPSAC '97. Proceedings., The Twenty-First Annual International*. 1997.

[51] L. Logrippo, "Special issue on feature interactions in telecommunications software." *Comput. Networks and ISDN Systems*, vol. 30, no. 15, 1998.

[52] L. Lorentsen, A.-P. Tuovinen, and J. Xu, "Modelling Feature Interaction Patterns in Nokia Mobila Phones using Coloured Petri Nets," in *23th International Conference on Application and Theory of Petri Nets*. Adelaide, Australia, Springer-Verlag Berlin Heidelberg, 2002.

[53] Y. Lu, G. Wei, and T.-Y. Cheung, "Managing feature interactions in telecommunications systems by Temporal Colored Petri nets." in *Proceedings of the Seventh IEEE International Conference on Engineering of Complex Computer Systems, 2001.* Skovde, Sweden, 2001.

[54] A. Metzger, "Feature interactions in embedded control systems." *Computer Networks*, vol. 45, no. 5, pp. 625–644, 2004.

[55] A. Metzger and C. Webel, "Feature Interaction Detection in Building Control Systems by Means of a Formal Product Model." in *Feature Interactions in Telecommunications and Software Systems VII*. Ottawa, Canada, IO Press, 2003.

[56] E.T. Mueller, "Event calculus and temporal action logics compared." *Artificial Intelligence*, vol. 170, no. 11, pp. 1017–1029, 2006.

[57] M. Nakamura, H. Igaki, and K.-i. Matsumoto, "Feature Interactions in Integrated Services of Networked Home Appliances: An Object Oriented Approach." in *8th International Conference on Feature Interactions in Telecommunications and Software Systems*. Leicester, UK, 2004.

[58] M. Nakamura, T. Kikuno, J. Hassine, and L. Logrippo, "Feature Interaction Filtering with Use Case Maps at Requirements Stage," in *Feature Interactions in Telecommunications and Software Systems VI*, M. Calder and E. Magill, Editors. IOS Press, 2000.

[59] M. Nakamura, P. Leelaprute, and T. Kikuno, "Deriving Interaction-Prone Scenarios in Feature Interaction Filtering with Use Case Maps." in *Proceedings of the Seventh IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS'02)*. 2002.

[60] S.R. Palmer and J.M. Felsing, *A Practical Guide to Feature-Driven Development*. Pearson Education, 2002.

[61] C. Phillips and L. Swiler, "A graph-based system for network-vulnerability analysis." *Proceedings of the 1998 workshop on New security paradigms*, pp. 71–79, 1998.

[62] K.P. Pomakis and J.M. Atlee, "Reachability analysis of feature interactions: a progress report." in *Proceedings of the 1996 ACM SIGSOFT international symposium on Software testing and analysis*. San Diego, California, United States: ACM Press New York, NY, USA. 1996.

[63] C. Ramakrishnan and R. Sekar, *Model-based analysis of configuration vulnerabilities.* Intrusion Detection, 2002.

[64] S. Reiff-Marganiec, "Policies: Giving Users Control over Calls," in *Agents, Objects and Features*, M.D. Ryan, J.-J.C. Meyer, and H.-D. Ehrlich, Ed. Springer Verlag, Berlin. pp. 189–208, 2004.

[65] S. Reiff-Marganiec and M.D. Ryan, *Feature Interactions in Telecommunications and Software Systems VIII.* Amsterdam, The Netherlands, IOS Press, 2005.

[66] S. Reiff-Marganiec and M.D. Ryan, "Guest Editorial." *Journal of Computer Networks: Special Issue on Feature Interaction*, vol. 51, no. 2, pp. 357–358, 2007.

[67] S. Reiff-Marganiec and K.J. Turner, "Feature Interaction in Policies." *Comput. Networks: The International Journal of Computer and Telecommunications Networking*, vol. 45, no. 5, pp. 569–584, 2004.

[68] W.N. Robinson, S.D. Pawlowski, and V. Volkov, "Requirements Interaction Management." *ACM Computi. Surv.*, vol. 35, no. 2, pp. 132–190, 2003.

[69] M. Shanahan, "The Event Calculus Explained," in *Lecture Notes in Computer Science*. Springer: Berlin / Heidelberg. p. 409, 1999.

[70] M. Shehata, A. Eberlein, and A.O. Fapojuwo, "A taxonomy for identifying requirement interactions in software systems." *Journal of Computer Networks: Special Issue on Feature Interaction*, vol. 51, no. 2, pp. 398–425, 2007.

[71] S. Siddiqi and J.M. Atlee, "A hybrid model for specifying features and detecting interactions." *Comput. Networks*, vol. 32, no. 4, pp. 471–485, 2000.

[72] G. Sindre and A.L. Opdahl, "Eliciting security requirements with misuse cases." *Journal of Requirements Engineering*, vo. 10, no. 1, pp. 34–44, 2005.

[73] G. Spanoudakis and K. Mahbub, "Non Intrusive Monitoring of Service Based Systems." *International Journal of Cooperative Information Systems*, vol. 15, no. 3, pp. 325–358, 2006.

[74] R. Telang and S. Wattal, "An Empirical Analysis of the Impact of Software Vulnerability Announcements on Firm Stock Price." *Software Engineering, IEEE Transactions* on, vol. 33, no. 8, pp. 544–557, 2007.

[75] S. Thiel, S. Ferber, T. Fischer, A. Hein, and M. Schlick, "A Case Study in Applying a Product Line Approach for Car Periphery Supervision Systems," in *Proceedings of In-Vehicle Software 2001 (SP-1587)*. Detroit, Michigan, USA. 2001.

[76] C.R. Turner, A. Fuggetta, L. Lavazza, and A.L. Wolf, "A Conceptual basis for feature engineering." *The Journal of Systems and Software*, vol. 49, no. 1, pp. 3–15, 1999.

[77] K.J. Turner, "Formalising the Chisel Feature Notation," in *Proceedings of the Feature Interactions in Telecommunications Networks VI*, M.H. Calder and E.H. Magill, Ed. IOS Press Amsterdam, Amsterdam. pp. 241–256, 2000.

[78] K.J. Turner and L. Blair, "Policies and conflicts in call control." *Journal of Computer Networks: Special Issue on Feature Interaction*, vol.51, no. 2, pp. 496–514, 2007.

[79] K.J. Turner, E.H. Magill, and D.J. Marples, *Service Provision*. Wiley Series in Communications Networking &

Distributed Systems, ed. D. Hutchison. John Wiley & Sons, Ltd. 2004.

[80] S. Uchitel and M. Chechik, "Merging Partial Behavioural Models." in *ACM International Symposium on Foundations of Software Engineering (FSE'04)*. Newport Beach, 2004.

[81] M. Weiss, *Detecting Feature Interactions in Web Services*. 2003, Carleton University, Ottawa, Canada.

[82] M. Weiss, "Feature Interactions in Web Services." in *Feature Interactions in Telecommunications and Software Systems VII, June 11-13, 2003*. Ottawa, 2003.

[83] M. Weiss, B. Esfandiari, and Y. Luo, "Towards a Classification of Web Service Feature Interactions." in *Proceedings Third International Conference Service-Oriented Computing - ICSOC 2005*.Amsterdam, The Netherlands: Springer Berlin/Heidelberg. 2005.

[84] M., Weiss, B. Esfandiari, and Y. Luo, "Towards a classification of web service feature interactions." *Journal of Computer Networks: Special Issue on Feature Interaction*, vol. 51, no. 2, pp. 359–381, 2007.

[85] X. Wu and H. Schulzrinne, "Handling feature interactions in the language for end system services." *Journal of Computer Networks: Special Issue on Feature Interaction*, vol. 51, no. 2, pp. 515–535, 2007.

[86] T. Yokogawa, T. Tsuchiya, M. Nakamura, and T. Kikuno, "Feature Interaction Detection by Bounded Model Checking." *IEICE Transactions on Information and Systems 2003*, vol. E86-D, no. 12, pp. 2579–2587, 2003.

[87] P.S. Yu and D.M. Dias, "Performance analysis of concurrency control using locking with deferred blocking." *Software Engineering, IEEE Transactions on*, vol. 19, no. 10, pp. 982–996, 1993.

[88] P. Zave, "Requirements for Evolving Systems: A Telecommunications Perspective." in *Fifth IEEE International Symposium on Requirements Engineering (RE '01), 2001*. IEEE Computer Society, 2001.

[89] P. Zave and M. Jackson, "Conjunction as composition." *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 2, no. 4, pp. 379–411, 1993.

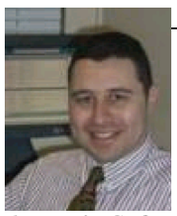
[90] P. Zave and M. Jackson, "Four dark corners of requirements engineering." *ACM Trans. Softw. Eng. Methodol. (TOSEM)*, vol. 6, no. 1, pp. 1–30, 1997.

Armstrong NHLABATSI

Armstrong NHLABATSI is a PhD student at The Open University. His research interests include feature composition and managing requirements inconsistency in feature-based applications. His PhD is focussed on the run-time management of feature interactions. He holds an MSc in Software Engineering from the University of the West of England and a B. Eng in Electronic Engineering from the University of Swaziland.

Robin LANEY

Robin LANEY is a Senior Lecturer in the Department of Computing at the Open University. His research interests include requirements engineering, flexible approaches to software architecture, and music computing. His research mission is to focus on activities that bridge the gap between theoretical advances and the experience and problems of working software practitioners, in both directions. He has industrial experience as a software engineer working on programming language technology and graphics. He holds a First Class Honours BSc in Microcomputers and Applications from Westfield College, University of London, and a PhD in Computing from King's College, University of London.



Bashar NUSEIBEH

Bashar NUSEIBEH is Professor and Director of Research in Computing at The Open University, and a Visiting Professor at Imperial College London and the National Institute of Informatics, Japan. He received his PhD degree in Software Engineering from Imperial College London in 1994. His research interests are in software requirements engineering and design, particularly applied to the development of dependable, mission-critical systems. Professor Nuseibeh is Editor-in-Chief of the Automated Software Engineering Journal, Chair of IFIP Working Group 2.9 on Requirements Engineering, and Chair of the Steering Committee of the International Conference on Software Engineering. He received a number of research and service awards, and is an Automated Software Engineering Fellow, a Fellow of the British Computer Society and the Institution of Engineering and Technology, and is a Chartered Engineer.