ACM Copyright Notice

Published in: ***Proceedings of International Software Product Line Conference (SPLC'14),*** September 2014

"Behaviour Interactions Among Product-Line Features"

Cite as:

Pourya Shaker and Joanne M. Atlee. 2014. Behaviour interactions among product-line features. In Proceedings of the 18th International Software Product Line Conference - Volume 1 (SPLC '14), Stefania Gnesi, Patrick Heymans, Julia Rubin, Krzysztof Czarnecki, Deepak Dhungana, and Alessandro Fantechi (Eds.), Vol. 1. ACM, New York, NY, USA, 242-246.

BibTex:

@inproceedings{Shaker:2014:BIP:2648511.2648538,
 author = {Shaker, Pourya and Atlee, Joanne M.},
 title = {Behaviour Interactions Among Product-line Features},
 booktitle = {Proceedings of the 18th International Software Product Line Conference - Volume 1},
 series = {SPLC '14},
 year = {2014},
 pages = {242--246}
}

DOI: https://doi.org/10.1145/2648511.2648538

Behaviour Interactions Among Product-Line Features

Pourya Shaker
University of Waterloo
p2shaker@uwaterloo.ca

Joanne M. Atlee
University of Waterloo
jmatlee@uwaterloo.ca

ABSTRACT

A software product line (SPL) is often constructed as a set of features, such that individual products can be assembled from a set of common features and a selection of optional features. Although features are conceptualized, developed, and evolved as separate concerns, it is often the case that, in practice, they interfere with each other – called a feature interaction. In this paper, we precisely define what it means for one feature to have a *behaviour interaction* with another feature, where the behaviour of one feature is affected by the presence of another feature. Specifically, we use a form of bisimilarity to define when the behaviour of a feature in isolation differs from its behaviour in the presence of an interacting feature. We also consider the case where features are modelled in a language that allows the specification of intended interactions, and we adapt our use of bisimilarity to provide a formal definition for unintended behaviour interactions.

Categories and Subject Descriptors

D.2 [**Software Engineering**]: Miscellaneous

Keywords

feature interactions, product lines, bisimulation

1. INTRODUCTION

Software product-line engineering (SPLE) is an increasingly popular approach to software development in which processes and practices are geared towards creating and managing a *family* of related products (e.g., smart phones, automobiles). Variability among products is characterized in terms of features, where a *feature* is a unit of functionality or added value. A software product line (SPL) includes a repository of mandatory and optional features, and individual products are derived by selecting among and integrating features from this feature set. The downside of SPLE is that, although features are conceptualized, developed, managed, and evolved as separate concerns, they often interfere with each other. In general, a *feature interaction* occurs whenever features influence one another in determining the overall system behaviour [17]. Feature interactions can manifest themselves in different ways. In the simplest cases, the actions of interacting features may conflict with each other or may violate a desired global invariant. For example, automotive features Cruise Control and Anti-lock Braking System may issue conflicting actions over the automobile's acceleration. Most of the early work on detecting feature interactions focused on interactions that manifest themselves as logical inconsistencies, such as conflicts, nondeterminism, deadlock, invariant violation, or satisfiability [4, 10, 7, 9, 14].

We are interested in the more enigmatic class of *behaviour interactions*, which are representative of how feature interactions result in emergent behaviours that cannot be attributed to any of the participating features. Specifically, a feature is developed and verified to be correct in isolation, but is found to *behave differently* when combined with other features. In this paper, we provide a precise definition of a behaviour interaction in terms of a violation of bisimilarity [13] between the behaviours of a feature in isolation and the behaviours of the feature when integrated with another (interacting) feature.

Some feature interactions are by design. For example, advanced Cruise Control features in automotive software *intentionally* override the behaviours of basic Cruise Control. A key complaint of feature-interaction definitions is that they do not distinguish between intended and unintended interactions. Analyses that use such definitions report a potentially large mix of intended and unintended interactions, leaving the user to sift through the reports looking for the subset of interactions that need to be addressed. As a second contribution, we provide a second definition of behaviour interaction that tolerates intended interactions: feature behaviour models specify intended interactions, and the definition of bisimilarity is weakened to admit (specified) intended interactions and reject unintended interactions. To our knowledge, the definition of feature interaction presented in this paper is the first definition that distinguishes between intended and unintended behaviour interactions.

The rest of this paper is organized as follows. Section 2 describes feature and SPL behaviour models. In Section 3, we present our use of bisimilarity to define general behaviour interactions, and Section 4 presents our amended definition that accommodates intended interactions. Section 5 states possible steps for applying our definitions in practice. Section 6 discusses related work, and we conclude in Section 7.

SPLC '14 Florence, Italy

2. BEHAVIOUR MODEL

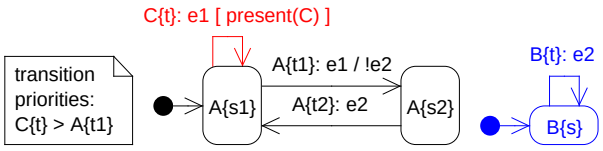

Figure 1: Behaviour model for an SPL *ExSPL* comprised of features A (black), B (blue) and C (red).

In this paper, the behaviours of features are specified as state machines and state-machine fragments. As a pedagogical example, Figure 1 shows the behaviour model for an SPL *ExSPL* with two mandatory features A and B and one optional feature C. We show only the SPL model that results from composing features A, B, and C. However, originally, features A and B are modelled as state machines and feature C is modelled as a fragment that extends feature A.

The states and transitions of a state machine have qualified names of the form $F\{n\}$, which specify the feature F to which a state or transition pertains. A state-machine transition has a label of the form $F\{t\} : te\ [gc]\ /\ a_1, \cdots, a_n$ where t is the name of the transition, te is a triggering event, gc is an optional guard condition, and $a_1 \cdots a_n$ are concurrent actions. An action is of the form $!e$ and specifies the generation of an event e.

The purpose of a new feature may (in part) be to modify the behaviours of an existing feature. In other words, a new feature can have *intended interactions* with an existing feature. We model a feature Y's intended interaction with a feature X as follows:

- *Transition priorities*: feature Y introduces a transition $Y\{t\}$ that takes priority over a transition $X\{t\}$ of feature X. The transition priority is denoted as $Y\{t\} > X\{t\}$.
- *Causing or preventing state changes*: feature Y introduces a transition that intentionally increases or decreases the conditions under which a feature X's state $X\{s\}$ is entered or exited. If a Y transition enters (or exits) state $X\{s\}$, then the transitions of X that are enabled by state $X\{s\}$ will execute under more (or fewer) conditions. If a Y transition takes priority over a transition that enters (or exits) state $X\{s\}$, then the transitions of X that are enabled by state $X\{s\}$ will execute under fewer (or more) conditions.

In the example of Figure 1, feature C has an intended interaction with feature A, which as modelled as follows: C's transition $C\{t\}$ takes priority over (eliminates) A's transition $A\{t1\}$. In doing so, $C\{t\}$ reduces the conditions under which feature A reaches state $A\{s2\}$ and prevents transition $A\{t2\}$ from ever executing.

The models of individual feature behaviours are composed into an SPL behaviour model: an integrated state-machine model representing the behaviours of all products of an SPL. In an SPL behaviour model, each transition of an optional feature F is guarded by a presence condition $present(F)$. The feature configuration of a product specifies which optional features are present in a product; if a feature is present, then its presence condition in the SPL model is true, which means that the feature's states and transitions are part of the product's behaviours.

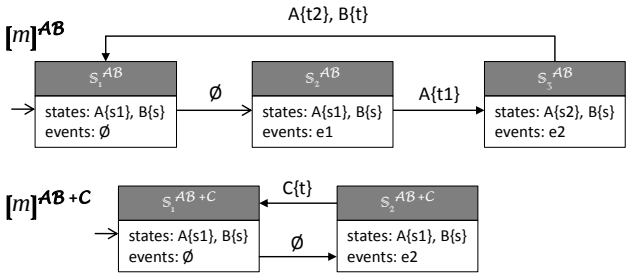

Figure 2: $[\![m]\!]^{\mathcal{AB}}$ and $[\![m]\!]^{\mathcal{AB}+\mathcal{C}}$ (initial states are denoted by a short incoming arrow)

The execution semantics of an SPL behaviour model m is given by the set of possible executions of all possible products derived by the SPL. The executions are represented by a a state-transition system $[\![m]\!]$. Each state of $[\![m]\!]$ consists of the set of executing machines and their current states, and the set of events to be processed by the machines in the next execution step. Each transition of $[\![m]\!]$ corresponds to a single execution step of the set of executing machines, and is labeled with the machines' transitions that execute in that step.

Behaviour interactions are not limited to features in the same product. They could be between features in different products operating in a shared environment (e.g., automotive features in two vehicles). Thus, we consider the execution semantics of a *product configuration* $\mathcal{P}$ comprising a set of SPL products, each with its own feature configuration. By extension, the executions of $\mathcal{P}$ are represented by a state-transition system $[\![m]\!]^{\mathcal{P}}$. Each state of $[\![m]\!]^{\mathcal{P}}$ consists of the execution states of all the machines that make up the products in $\mathcal{P}$. Each transition in $[\![m]\!]^{\mathcal{P}}$ is labelled with all of the machines' transitions in that execution step. From $\mathcal{P}$'s executions, we use projection to extract the behaviours of a particular feature F. Projection $[\![m]\!]^{\mathcal{P}|F}$ represents F's behaviours in $[\![m]\!]^{\mathcal{P}}$.

Example 1: Figure 2 shows the state-transition systems for two product configurations of *ExSPL*: the product configuration $\mathcal{AB}$ comprises a single product with features A and B, and the product configuration $\mathcal{AB}+\mathcal{C}$ results from adding feature C to the product in $\mathcal{AB}$. $[\![m]\!]^{\mathcal{AB}}$ executes transition $A\{t1\}$ in response to an environment-generated event $e1$, and then executes transitions $A\{t2\}$ and $B\{t1\}$ in response to event $e2$ generated by $A\{t1\}$. In contrast, $[\![m]\!]^{\mathcal{AB}+\mathcal{C}}$ executes transition $C\{t\}$, instead of $A\{t1\}$, in response to the environment event $e1$; $A\{t2\}$ and $B\{t1\}$ no longer execute due to the absence of event $e2$. The projection $[\![m]\!]^{\mathcal{AB}|A}$ replaces the label of each $[\![m]\!]^{\mathcal{AB}}$ transition with the subset of A transitions in the label. For example, the transition label $A\{t2\}, B\{t\}$ in $[\![m]\!]^{\mathcal{AB}}$ is replaced with the label $A\{t2\}$ in $[\![m]\!]^{\mathcal{AB}|A}$. Other projections are obtained in a similar manner.

3. BEHAVIOUR INTERACTIONS

The features of a product, or the features of different products, can modify one another's behaviours. A feature Y modifies the behaviours of another feature X by inhibiting or triggering X's behaviours. When Y modifies the behaviours of X, we say that Y has a *behaviour interaction* with X.

Behaviour interactions manifest themselves as follows. Let $\mathcal{P}$ be a minimal product configuration (set of products)

that includes feature X (and all features that feature X depends on). Suppose that a feature Y is added to some product in $\mathcal{P}$, resulting in the extended product configuration $\mathcal{P}+\mathcal{Y}$[1]. To determine whether Y has a behaviour interaction with X, we compare the executions of $\mathcal{P}$ and $\mathcal{P}+\mathcal{Y}$. Informally, a behaviour interaction is said to occur if X's behaviours in the executions of $\mathcal{P}+\mathcal{Y}$ are different from X's behaviours in the executions of $\mathcal{P}$.

The comparison of the executions of $\mathcal{P}$ and $\mathcal{P}+\mathcal{Y}$ is formally expressed as a *bisimilarity* check between $[\![m]\!]^{\mathcal{P}|X}$ and $[\![m]\!]^{\mathcal{P}+\mathcal{Y}|X}$. Whether $[\![m]\!]^{\mathcal{P}|X}$ and $[\![m]\!]^{\mathcal{P}+\mathcal{Y}|X}$ are bisimilar is determined by playing a *matching game*. The game starts with the transition systems in their initial states. In each round of the game, either transition system can take a transition, hereafter called a *move*. For all possible next moves in one system, there must exist a *matching move* in the other system, and vice versa. Recall that a move represents the concurrent execution of a set of machine transitions. Two moves match if the transitions that execute in the two moves are equal. In Example 1, the move $\mathbb{s}_1^{\mathcal{AB}|A} \xrightarrow{\emptyset} \mathbb{s}_2^{\mathcal{AB}|A}$ in $[\![m]\!]^{AB|A}$ matches the move $\mathbb{s}_1^{\mathcal{AB}+\mathcal{C}|A} \xrightarrow{\emptyset} \mathbb{s}_2^{\mathcal{AB}+\mathcal{C}|A}$ in $[\![m]\!]^{\mathcal{AB}+\mathcal{C}|A}$ since the sets of A transitions executed in the two moves are the same (are both empty).

If the two transition systems can match each other's moves in all rounds of the matching game then the same is a *winning game* and the state-transition systems are bisimilar. A *bisimulation relation* relates pairs of states, one from each state-transition system, in each round of a winning game.

Definition 1. $[\![m]\!]^{\mathcal{P}|X}$ is bisimilar to $[\![m]\!]^{\mathcal{P}+\mathcal{Y}|X}$ if and only if there exists a bisimulation relation BSR between the states of $[\![m]\!]^{\mathcal{P}|X}$ and $[\![m]\!]^{\mathcal{P}+\mathcal{Y}|X}$ that specifies a winning game:

1. The pair of initial states of $[\![m]\!]^{\mathcal{P}|X}$ and $[\![m]\!]^{\mathcal{P}+\mathcal{Y}|X}$ are related by BSR.
2. Every move of $[\![m]\!]^{\mathcal{P}|X}$ is matched by a move of $[\![m]\!]^{\mathcal{P}+\mathcal{Y}|X}$. That is, if states $\mathbb{s}^{\mathcal{P}|X}$ and $\mathbb{s}^{\mathcal{P}+\mathcal{Y}|X}$ are related by BSR, then (a) every move out of $\mathbb{s}^{\mathcal{P}|X}$ is matched by a move out of $\mathbb{s}^{\mathcal{P}+\mathcal{Y}|X}$, and (b) the destination states of the matching moves are also related by BSR.
3. Every move of $[\![m]\!]^{\mathcal{P}+\mathcal{Y}|X}$ is matched by a move of $[\![m]\!]^{\mathcal{P}|X}$. That is, if states $\mathbb{s}^{\mathcal{P}|X}$ and $\mathbb{s}^{\mathcal{P}+\mathcal{Y}|X}$ are related by BSR, then (a) every move out of $\mathbb{s}^{\mathcal{P}+\mathcal{Y}|X}$ is matched by a move out of $\mathbb{s}^{\mathcal{P}|X}$, and (b) the destination states of the matching moves are also related by BSR.

A behaviour interaction, then, can be defined in terms of a violation of bisimilarity.

Definition 2. Let m be an SPL behaviour model, $\mathcal{P}$ be a product configuration, and $\mathcal{P}+\mathcal{Y}$ be the product configuration that results from adding feature Y to $\mathcal{P}$. Y has a behaviour interaction with a feature X in $\mathcal{P}$ if and only if $[\![m]\!]^{\mathcal{P}|X}$ is not bisimilar to $[\![m]\!]^{\mathcal{P}+\mathcal{Y}|X}$.

Consider the product configurations $\mathcal{AB}$ and $\mathcal{AB}+\mathcal{C}$ from Example 1. The following examples determine whether the new feature C has behaviour interactions with the existing features[2] by checking bisimilarity between projections over the transitions systems $[\![m]\!]^{\mathcal{AB}}$ and $[\![m]\!]^{\mathcal{AB}+\mathcal{C}}$ in Figure 2.

Example 2: C has a behaviour interaction with A because $[\![m]\!]^{\mathcal{AB}|A}$ is not bisimilar to $[\![m]\!]^{\mathcal{AB}+\mathcal{C}|A}$. Starting in the initial states $\mathbb{s}_1^{\mathcal{AB}|A}$ and $\mathbb{s}_1^{\mathcal{AB}+\mathcal{C}|A}$, $[\![m]\!]^{\mathcal{AB}+\mathcal{C}|A}$ can match $[\![m]\!]^{\mathcal{AB}|A}$'s move of executing no A transitions. However, in the resulting states $\mathbb{s}_2^{\mathcal{AB}|A}$ and $\mathbb{s}_2^{\mathcal{AB}+\mathcal{C}|A}$, $[\![m]\!]^{\mathcal{AB}+\mathcal{C}|A}$ cannot match $[\![m]\!]^{\mathcal{AB}|A}$'s move of executing transition $A\{t1\}$.

Example 3: C has a behaviour interaction with B because $[\![m]\!]^{\mathcal{AB}|B}$ is not bisimilar to $[\![m]\!]^{\mathcal{AB}+\mathcal{C}|B}$. Starting in the initial states $\mathbb{s}_1^{\mathcal{AB}|B}$ and $\mathbb{s}_1^{\mathcal{AB}+\mathcal{C}|B}$, $[\![m]\!]^{\mathcal{AB}+\mathcal{C}|B}$ can match $[\![m]\!]^{\mathcal{AB}|B}$'s move of executing no B transitions. The same is true in the resulting states $\mathbb{s}_2^{\mathcal{AB}|B}$ and $\mathbb{s}_2^{\mathcal{AB}+\mathcal{C}|B}$. However, in the following round in states $\mathbb{s}_3^{\mathcal{AB}|B}$ and $\mathbb{s}_1^{\mathcal{AB}+\mathcal{C}|B}$, $[\![m]\!]^{\mathcal{AB}+\mathcal{C}|B}$ cannot match $[\![m]\!]^{\mathcal{AB}|B}$'s move of executing transition $B\{t\}$.

C's behaviour interaction with A is intended: in $[\![m]\!]^{\mathcal{AB}+\mathcal{C}|A}$'s move, transition $C\{t\}$ takes priority over transition $A\{t1\}$. Whereas, C's behaviour interaction with B is unintended: in $[\![m]\!]^{\mathcal{AB}+\mathcal{C}|B}$'s execution, transition $C\{t\}$ takes priority over transition $A\{t1\}$; this unintentionally disables transition $B\{t\}$ in $[\![m]\!]^{\mathcal{AB}+\mathcal{C}|B}$'s next round move, because $B\{t\}$ can be triggered only by $A\{t1\}$'s generation of event $e2$. Hence, the general definition of behaviour interactions includes C's intended interaction with A and C's unintended interaction with B.

4. UNINTENDED BEHAVIOUR INTERACTIONS

In the general definition given in Section 3, an added feature Y has a behaviour interaction with an existing feature X in product configuration $\mathcal{P}$ if the state-transition systems $[\![m]\!]^{\mathcal{P}}$ and $[\![m]\!]^{\mathcal{P}+\mathcal{Y}}$ cannot match one another's moves in all rounds of the matching game. However, two associated moves may fail to match *by design* because there are intended interactions. An intended interaction by feature Y will trigger or inhibit the transitions of X in $[\![m]\!]^{\mathcal{P}+\mathcal{Y}}$'s moves. Such interactions affect not only X's immediate behaviours, but can have long lasting effects if, because of a triggered or inhibited transition, a feature is in a different state and thus has different future behaviours. In this work, we presume that if an intended interaction between features Y and X result in different future behaviours in X, then the resulting feature behaviours are also intended – although this presumption may be optimistic. Thus, there are two classes of intended interactions to consider:

1. *Intended match:* A transition $X\{t\}$ in the current $[\![m]\!]^{\mathcal{P}+\mathcal{Y}}$ move is inhibited by a higher-priority transition of Y. In order to ignore such intended interactions, we weaken the notion of a match between moves to permit differences caused by transition priorities.
2. *Weakened winning game:* A transition $X\{t\}$ of X is inhibited in a *future* $[\![m]\!]^{\mathcal{P}+\mathcal{Y}}$ move, because a transition of Y in the current $[\![m]\!]^{\mathcal{P}+\mathcal{Y}}$ move intentionally triggers the exit or prevents the entry of an X state on which

[1]Feature Y could be added to the product in $\mathcal{P}$ that includes feature X or could be added to another product in $\mathcal{P}$.

[2]In our example, the product configuration that includes both A and B is a minimal product configuration because A and B are both mandatory features.

$X\{t\}$'s enabledness depends. If the Y transition intentionally triggers the exit or prevents the entry of the X state, then transition $X\{t\}$ may appear in a future $[\![m]\!]^{\mathcal{P}}$ move, but not appear in any associated future $[\![m]\!]^{\mathcal{P}+\mathcal{Y}}$ move. Analogously, if the Y transition intentionally triggers the entry or prevents the exit of the X state, the transition $X\{t\}$ may appear in a future $[\![m]\!]^{\mathcal{P}+\mathcal{Y}}$ move but not in any associated $[\![m]\!]^{\mathcal{P}}$ move. In order to ignore such intended interactions, we weaken the notion of a winning game as follows: if in the course of a matching game $[\![m]\!]^{\mathcal{P}+\mathcal{Y}}$ performs a Y transition that intentionally triggers or inhibits the entry or exit of an X state, then the game is won. Considering these games to be won is equivalent to tolerating any future mismatches between the moves of $[\![m]\!]^{\mathcal{P}}$ and $[\![m]\!]^{\mathcal{P}+\mathcal{Y}}$.

We now provide a new definition for bisimilarity, which revises clauses 2(a) and 3(a) in Definition 1 to use the notion of an intended match between the moves of $[\![m]\!]^{\mathcal{P}|X}$ and $[\![m]\!]^{\mathcal{P}+\mathcal{Y}|X}$ as defined above; and revises clauses 2(b) and 3(b) to encode the weakened notion of a winning game as described above. The revised clauses are shown in red.

Definition 3. $[\![m]\!]^{\mathcal{P}|X}$ is intentionally bisimilar to $[\![m]\!]^{P+Y}$ if and only if there exists a *intentional-bisimulation relation* BSR between the states of $[\![m]\!]^{\mathcal{P}|X}$ and $[\![m]\!]^{\mathcal{P}+\mathcal{Y}|X}$ such that:

1. The pair of initial states of $[\![m]\!]^{\mathcal{P}|X}$ and $[\![m]\!]^{\mathcal{P}+\mathcal{Y}|X}$ are related by BSR.
2. Every move of $[\![m]\!]^{\mathcal{P}|X}$ is matched by a move of $[\![m]\!]^{\mathcal{P}+\mathcal{Y}|X}$: if states $\mathbb{s}^{\mathcal{P}|X}$ and $\mathbb{s}^{\mathcal{P}+\mathcal{Y}|X}$ are related by BSR, then (a) every move out of $\mathbb{s}^{\mathcal{P}|X}$ is intentionally matched by a move out of $\mathbb{s}^{\mathcal{P}+\mathcal{Y}|X}$; and (b) if the move of $[\![m]\!]^{\mathcal{P}+\mathcal{Y}|X}$ includes no Y transition that intentionally triggers or inhibits the entry or exit of an X state, then the destination states of the matching moves are also related by BSR.
3. Every move of $[\![m]\!]^{\mathcal{P}+\mathcal{Y}|X}$ is matched by a move of $[\![m]\!]^{\mathcal{P}|X}$: if states $\mathbb{s}^{\mathcal{P}|X}$ and $\mathbb{s}^{\mathcal{P}+\mathcal{Y}|X}$ are related by BSR, then (a) every move out of $\mathbb{s}^{\mathcal{P}+\mathcal{Y}|X}$ is intentionally matched by a move out of $\mathbb{s}^{\mathcal{P}|X}$; and (b) if the move of $[\![m]\!]^{\mathcal{P}+\mathcal{Y}|X}$ includes no Y transition that intentionally triggers or inhibits the entry or exit of an X state, then the destination states of the matching moves are also related by BSR.

We can now formally define unintended behaviour interactions, in terms of a violation of bisimilarity as defined above.

Definition 4. Let m be an SPL behaviour model, $\mathcal{P}$ be a product configuration, and $\mathcal{P}+\mathcal{Y}$ be the product configuration that results from adding feature Y to $\mathcal{P}$. Y has an unintended behaviour interaction with a feature X in $\mathcal{P}$ if and only if $[\![m]\!]^{\mathcal{P}|X}$ is not intentionally bisimilar to $[\![m]\!]^{\mathcal{P}+\mathcal{Y}|X}$.

Consider the product configurations $\mathcal{AB}$ and $\mathcal{AB}+\mathcal{C}$ from Example 1. The following examples revisit the question of whether C has behaviour interactions with A and B, using the revised definition of bisimilarity.

Example 4: C's behaviour interaction with A in Example 2 is tolerated as being intended because $[\![m]\!]^{\mathcal{AB}|A}$ is intentionally bisimilar to $[\![m]\!]^{\mathcal{AB}+\mathcal{C}|A}$: Recall from Example 2 that in the third round of the game, $[\![m]\!]^{\mathcal{AB}+\mathcal{C}|A}$'s move of executing transition $C\{t\}$ does not match $[\![m]\!]^{\mathcal{AB}|A}$'s move of executing transition $A\{t1\}$. However, the moves *intentionally match* because $C\{t\}$ takes priority over $A\{t1\}$. Furthermore, since the transition $C\{t\}$ prevents entry to state $A\{s2\}$ (the destination state of the preempted transition $A\{t1\}$), future mismatches are tolerated and the matching game is won.

Example 5: C has a behaviour interaction with B because $[\![m]\!]^{\mathcal{AB}|B}$ is not bisimilar to $[\![m]\!]^{\mathcal{AB}+\mathcal{C}|B}$. Recall from Example 3 that in the third round of the game, the two transition systems cannot match one another's moves. However, the mismatch does not occur because of constructs for modelling intended interactions. As explained in Example 3, the mismatch occurs because $[\![m]\!]^{\mathcal{AB}+\mathcal{C}|B}$'s previous move preempts the generation of event $e2$ that would trigger transition $B\{t\}$.

Hence, the revised definition of behaviour interactions tolerates C's intended interaction with A and reports C's unintended interaction with B.

5. ROADMAP TO PRACTICE

This section proposes some steps for putting our definitions of behaviour interactions to practical use.

- To simplify presentation, the definitions in this paper are given for canonical state-machine models. Details about how these definitions apply to a richer SPL modelling language (i.e., one that includes a rich data model, and richer constructs for expressing actions, events, and intended interactions in state machines), including a formalization of the language's execution semantics and corresponding bisimulation relations are given in [16]. [16] also includes a formalization of other prominent feature-interaction types for state-machine models, such as conflicting actions, nondeterminism, deadlock, and looping. However, behaviour interactions are more general and subsume these feature-interaction types.
- Our behaviour interaction definitions could form the basis for an analysis technique that detects such interactions. A natural choice for analysis would be to adapt well-known algorithms for checking bisimilarity [1] – to incorporate the notion of an intended match and the weakened notion of a winning game, so that the analysis reports only unintended interactions. A technical challenge is that the size of the state space can grow quickly with the size of the product configurations being analyzed as well as the (modelling-language dependent) size of auxiliary data. One approach to address this challenge would be to place bounds on the product-configuration and data sizes, or on the length of the executions considered in analysis; however, the impact of such bounds on the effectiveness of analysis need to be evaluated.
- A systematic process should be developed for applying such an analysis to the features of an SPL, such that coverage of behaviour interactions is guaranteed. One approach is to check behaviour interactions between each pair X and Y of features by comparing the executions of a minimal product configuration that includes X to that obtained by adding Y this product configuration. However, other work shows that pairwise feature-interaction analysis is not complete [3]. One idea is to extend the minimal product configuration of X with other features

that are suspected to cause feature interactions in combination with X and Y, for example, because they operate on the same environment phenomena.

- The analysis and process developed for detecting behaviour interactions should be evaluated using real-world cases studies. Evaluation should focus both on computational efficiency as well as on the utility of analysis results (e.g., number of false positives). Evaluation may identify avenues for optimizing the process and analysis, to improve scalability and the utility of results.

6. RELATED WORK

There is extensive work on formally defining how specific classes of unintended feature interactions manifest themselves in behavioural models, such as logical inconsistency [4], the violation of correctness properties [8], conflicting actions [9], nondeterminism [12], and deadlock [10]. LaPorta et al. [11] provide a formal definition of feature interactions based on trace equivalence that is similar to our notion of behaviour interactions: a set of features F are said to interact with another set of features E if the projected set of execution traces of F and $F \cup E$ differ. Broy [5] defines a theory for specifying a system's interface behaviour (possible streams of IO messages) as an integrated state machine, where the behaviours of individual features are derived by projection. Similar to the work by LaPorta et al., the independence (or lack of behaviour interactions) between features is formalized as a form of trace equivalence. We chose bisimilarity over trace equivalence to define feature interactions, because bisimilarity is more sensitive to nondeterminism (possible both within and between features) than trace equivalence is. To our knowledge, there is no general definition of behaviour interactions that distinguishes between intended and unintended interactions.

The aspect-oriented software development community has also studied interactions among separate concerns developed as aspects. Typical definitions given for aspect interactions are aspects advising overlapping parts of a base program or other aspects [6] and aspects violating correctness properties of other aspects [2]. However, such definitions do not correspond to the general case of behaviour interactions. In a more closely related approach, Rinard et al. [15] classify interactions among aspects and the base program based on their direct and indirect effects on one another's control flow. Our approach differs from that of Rinard et al. in the artefact over which interactions are defined (state-machine models vs. code) and the technique used to define interactions (bisimilarity vs. pointer and escape analysis).

7. CONCLUSION AND FUTURE WORK

We have defined how a key class of feature interactions, called behaviour interactions, manifest themselves in behaviour models of SPL requirements. Informally, a feature Y has a behaviour interaction with a feature X, in the same or in a different product, if Y modifies (e.g., triggers, blocks) X's behaviours. The paper's contributions are (1) a definition of behaviour interaction in terms a difference between the projection of a feature X in some product configuration and X's projection in the same product configuration enhanced with feature Y; and (2) a formal definition of unintended behaviour interaction in terms of the violation of intentional-bisimilarity. The latter definition could be the basis for an analysis that reports only unintended interactions. For future work, we plan to automate the detection of behaviour interactions.

8. ACKNOWLEDGMENTS

We would like to thank Andrzej Wąsowski for his early feedback on this work.

9. REFERENCES

[1] L. Aceto, A. Ingolfsdottir, and J. Srba. *Advanced Topics in Bisimulation and Coinduction*, chapter The Algorithmics of Bisimilarity, pages 100–172. 2011.

[2] Z. Altahat and T. Elrad. Detection and verification of semantic interaction in AOSD. In *ITNG*, pages 807–812, 2009.

[3] S. Apel, S. S. Kolesnikov, N. Siegmund, C. Kästner, and B. Garvin. Exploring feature interactions in the wild: the new feature-interaction challenge. In *FOSD@GPCE*, pages 1–8, 2013.

[4] J. Blom, B. Jonsson, and L. Kempe. Using temporal logic for modular specification of telephone services. In *FIW*, pages 197–216, 1994.

[5] M. Broy. Multifunctional software systems: Structured modeling and specification of functional requirements. *Sci. Comput. Program.*, 75(12):1193–1214, 2010.

[6] R. Douence, P. Fradet, and M. Südholt. A framework for the detection and resolution of aspect interactions. In *GPCE*, pages 173–188, 2002.

[7] A. P. Felty and K. S. Namjoshi. Feature specification and automated conflict detection. *TOSEM*, 12(1):3–27, 2003.

[8] A. Gammelgaard and J. Kristensen. Interaction detection, a logical approach. In *Feature Interactions in Telecommunications Systems*, pages 178–196, 1994.

[9] R. J. Hall. Feature combination and interaction detection via foreground/background models. *Comput. Netw.*, 32(4):449–469, 2000.

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

[11] T. F. LaPorta, D. Lee, Y.-J. Lin, and M. Yannakakis. Protocol feature interactions. In *FORTE*, volume 135, pages 59–74, 1998.

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

[13] D. Park. Concurrency and automata on infinite sequences. In *Theoretical Computer Science*, pages 167–183, 1981.

[14] M. Plath and M. Ryan. Feature integration using a feature construct. *Sci. Comput. Program.*, 41(1):53–84, 2001.

[15] M. Rinard, A. Salcianu, and S. Bugrara. A classification system and analysis for aspect-oriented programs. In *FSE*, pages 147–158, 2004.

[16] P. Shaker. A feature-oriented modelling language and a feature-interaction taxonomy for product-line requirements. Ph.D. Thesis, 2013.

[17] P. Zave. Requirements for evolving systems: a telecommunications perspective. In *RE*, pages 2–9, 2001.