Generic Feature Interaction Detection in Web Services

Ahmed Khoumsi¹

Department of Computer Science, King Saud University, Kingdom of Saudi Arabia
Ahmed.Khoumsi@USherbrooke.ca

Abstract.
Web service composition, which denotes the design of new WS by composing existing ones, has become a powerful Web service design approach. However, such a composition sometimes generates undesirable behaviors, which are referred to as feature interactions. The latter must be detected and resolved. In this paper, we develop a generic method to detect feature interactions in Web services during their design (from scratch or by composing existing WS). The method uses model-checking techniques to detect feature interactions which are specified by formal expressions called invariants. The detection method is generic in the sense that it may be applicable for the detection of any feature interaction that can be specified by invariants. We demonstrate the applicability of our feature interaction detection method in several examples.

Keywords. Web service composition; Feature interactions (FI); Off-line and generic FI detection; CRESS description; Model checking; Invariants.

1. Introduction
The great need of WS (WS) can benefit from the principle of reusability by composing existing WS in order to synthesize new WS. However, these compositions may generate problematic behaviors, which are commonly called Feature Interactions (FI). The FI problem has been studied in telecommunications services (or Telecom-services) since the 80s, and many solutions have emerged to detect and solve FI (e.g., [13]). A first idea that may come in mind is to adapt those solutions to the context of WS. However, this adaptation may be difficult and sometimes impossible. Let us explain it in a few words. Firstly, a WS can be any imaginable software system providing a service through the Web, while Telecom-services constitute a specific category of services which can generally be abstracted by some parameters, as it has been done in [4] and [10]. Secondly, composing Telecom-services generally means just running them in parallel, while composing WS means designing a new WS by composing existing WS, by applying a design methodology as it should in any well conducted software design project. Hence, composing WS and consequently managing their FI must be done off-line (i.e., at design phase). In other words, we think that it is unrealistic to compose WS and detect their FI by on-line approaches (i.e., at runtime) since a design phase is necessary.

To address the complexity of WS composition, several composition models have been developed, such as orchestration and choreography. Orchestration consists in using a specific WS which orchestrates several WS. Choreography is more complex than orchestration and can be seen as some kind of mutual orchestration. Many studies for composing WS and detecting FI among WS have been done, such as those presented in Section 2.

The structure and the main contributions of this paper are as follows: Section 2 presents some related work on composing WS and detecting their FI. Section 3 introduces the basic principle of our FI detection method, the formalism we have used to describe WS, and the notion of invariants which are used to specify FI formally. In Section 4, we propose a generic method for detecting FI in a WS during its design. By generic,
we mean to have a single FI detection method which is configurable by the category (s) of FI to detect, the latter being specified by invariants. Sections 5 and 6 present applications of our FI detection method to concrete examples. Section 5 presents the detection of an FI in an example of [16]. Section 6 demonstrates that our method detects all FI of the benchmark of [19]. Our FI detection has also been applied to detect several other FI such as those in examples of [2, 17], but we do not present them due to space limitation. Section 7 concludes the paper.

2. Related Work on Composing and Detecting FI in WS

A valuable contribution of [17, 19] is to have raised interest and awareness of researchers to the problem of WS composition and FI detection. Moreover, [19] presents a benchmark consisting of a series of examples of WS composition and WS FI which can be used to assess FI detection methods. Another contribution in raising interest of researchers can be found in [2], which contains a comparative study showing that FI in Telecom-services differ from FI in WS.

Significant work has been done in on-line WS composition and FI management, e.g. [20, 23, 24]. [20] contains an on-line FI detection method inspired from the Situation Calculus. [23] presents an on-the-fly approach to compose WS. [24] argues that WS composition can be done at-run-time and identifies some challenges and opportunities in FI detection and resolution.


On business model development, it is worth mentioning [7, 8] that propose an extension of the standard business model of [5] to support WS composition. The authors of [7, 8] go further in [9] by studying how WS can be categorized and assembled.

[15, 16] are particularly relevant since they contain a rigorous study of WS composition, where both theoretical, software-tooling and user-interfacing aspects are considered. The CRESS formalism is used which can be automatically translated into BPEL and LOTOS.

3. Preliminaries to our FI detection method

WS composition targets to design a new WS by composing existing WS, and FI detection targets to check whether a designed WS is FI prone. Before to present our FI detection procedure in Section 4, let us present the basic principle of our FI detection method and the formalisms used to describe WS and FI formally. The WS to be composed are denoted $S_1, \ldots, S_n$, and the new WS designed by composition is denoted $\text{Comp}(S_1, \ldots, S_n)$.

3.1. Basic principle of our FI detection method within a WS design environment

Our FI detection procedure is executed during the design of a new WS. FI detection is off-line and realized by analyzing a formal description of the new WS. If the new WS is being designed by composing several WS $S_1, \ldots, S_n$, the formal description of the new WS is constructed from formal descriptions of $S_1, \ldots, S_n$. If the new WS is being designed from scratch, its formal description will be constructed from
scratch. Hence, in any case, a formal description of the WS under design has to be constructed. FI detection is realized by analyzing the newly constructed description, using model checking techniques.

3.2. Suggested formalism to describe WS

We have selected the formalism CRESS because the author of [16] provides many convincing arguments that CRESS is particularly adequate for WS description, composition and analysis. Here are some of the advantages of CRESS cited in [16]: CRESS is high-level, graphical and intuitive. CRESS offers automated translation to formal specifications (e.g. LOTOS, SDL) as well as to implementations (e.g. BPEL, WSDL), which means that CRESS promotes automated verification and validation as well as automated implementation. By applying to CRESS the FI detection approach explained in Section 3.1, a CRESS description of the new WS is constructed from scratch or by composing CRESS descriptions of $S_1, ..., S_n$.

Due to space limitation, we do not present CRESS (the interested reader is invited to consult [16]). This is not a problem for the comprehension of our FI detection method, because its principle and applicability are not restricted to CRESS. The latter is suggested for its advantages, and not for its indispensability. Indeed, our FI detection method can be easily adapted to any other formalisms from which a reachability tree can be synthesized, e.g., automata, Petri nets.

3.3. Invariants to specify FI formally

FI detection is realized by verifying whether the CRESS description of the WS under design respects some properties which are formally expressed by some formulae called invariants. Therefore, FI detection consists in checking whether there exists a path of execution where invariants are violated. The detection procedure is generic, in the sense that it is intended to detect any FI that can be modeled by an invariant. Each invariant is formally specified by an expression that evaluates to true or false.

Consider for example the requirement: the WS must not contain a deadlock. This requirements can be modeled by using the invariant “$\text{NbEnabEvents} > 0$”, where $\text{NbEnabEvents}$ is a variable denoting the Number of enabled events. Intuitively, there is at least an executable event in any situation. Here is another example: the price including taxes is greater than the price excluding taxes. This requirement is modeled by the invariant “$x > y$”, where $x$ and $y$ are two variables representing the two prices respectively.

3.4. Examples of invariants

Here are examples of invariants expressed intuitively. For each example, we indicate the subsection where it will be used.

**Absence of deadlock**: the number of enabled events is positive (Section 6.1).

**Order completion/abortion**: in a virtual bookstore WS, when a book order is completed, the ordered book has been paid and delivered (Section 6.2). And when a book order is aborted, the ordered book has not been paid nor delivered (Section 6.3).

**Equivalent variables $x_1$ and $x_2$**: when two variables have the same semantics, they must be equal (Section 6.4).

**Profile Access**: when a user profile is accessed, the profile access policy is respected (Section 6.5).

**Availability**: in a Car Supplier WS, the supplier is always available (Section 6.6).
4. FI detection procedure

As explained in Section 3.1, our procedure for detecting FI in a WS is executed during the design of such a WS. The following sections describe the steps of our FI detection procedure.

4.1. Step 1: Constructing a CRESS-description of the WS under design

If the new WS is being designed by composing several WS $S_1$, $S_2$, ..., $S_n$, the CRESS descriptions of every $S_1$, ..., $S_n$ are inputs of the FI detection procedure. These CRESS descriptions are composed in order to construct a CRESS description of the new WS $\text{Comp}(S_1, ..., S_n)$. Composition can use orchestration, choreography or a mixture of them. Anyway, whatever composition method is used, this is a design phase which cannot be automated, and hence requires a human participation.

If the new WS is being designed from scratch, its CRESS-description is constructed from scratch. This case can be conceptually considered as a particular case of the above case by taking $n$ (number of composed WS) equal to zero. Hence, for simplicity of the presentation, this particular case is not considered in the following steps.

4.2. Step 2: Completing the description of the new WS by invariants

We need to complete the CRESS description of $\text{Comp}(S_1, ..., S_n)$ by invariants that specify formally the FI to be detected. We may think of a FI detection environment that contains by default a list of generic invariants that can be enabled or disabled by the designer. In addition to the generic invariants, the designer can specify some invariants that model specific FI. For example, the two invariants $\text{NbEnabEvents} > 0^*$ and $^*x > y^*$ can be considered as a generic and a specific invariants, respectively. All the selected invariants (generic, specific) are associated to the CRESS description of $\text{Comp}(S_1, ..., S_n)$, to obtain what we call completed description of $\text{Comp}(S_1, ..., S_n)$.

4.3. Step 3: Constructing the reachability tree and checking the invariants

The reachability tree of the completed description of $\text{Comp}(S_1, ..., S_n)$ is constructed, where the invariants defined in Step 2 are evaluated in each node during its construction. A non-respect of an invariant corresponds to the occurrence of a FI. Let FI-node denote a node where an invariant is not respected. Detecting so many FI as possible may be doubly time consuming, because it requires constructing many nodes and checking many invariants in each node. This temporal cost can be reduced by two numbers $N_{\text{INV}}$ and $N_{\text{FL}}$. The number of invariants is limited to $N_{\text{INV}}$, and the construction of the reachability tree is stopped before its completion if a number $N_{\text{FL}}$ of FI-nodes have been constructed. More generally, any technique in verification of formal expressions can be used. Much work has been done in verification, and we do not pretend to contribute specifically in this step. We think that our contribution is more global by the combination of several steps and concepts to obtain a method for detecting FI in WS.

4.4. Step 4: Sending comprehensive messages

In order to help designers to solve FI, a comprehensive message should be generated for each detected FI (i.e., non respected invariant). The message indicates in an intuitive way:

- the type of FI: i.e., the non-respected invariant;
- the executed path that has led to the FI.

5. Application to detect a “new” FI in the example of in [16]

In this section, we demonstrate our method in the example of [16]. The detected FI is in fact due to an error we have unconsciously made when rewriting a WS description of that example.
5.1. Example of [16]: Lender, Car Supplier and Car Broker

The example of [16] consists of three WS: a Lender, a Car Supplier, and a Car Broker. The Lender is designed by composing two WS: a Loan Approver and a Loan Assessor, and the Car Supplier is designed by composing two Car Dealers. Then, the Car Broker is designed by composing the Lender and the Car Supplier. We will not present the CRESS descriptions of Lender, Car Supplier and Car Broker which are given in [16]. Let us just introduce them intuitively:

Lender: It is invoked to approve or assess a loan of a given amount. The approver WS is first invoked if amount ≥ 10000, and the assessor WS is invoked otherwise (i.e., if amount < 10000). The approver evaluates the loan proposal, and then decides to refuse or approve it. In the latter case, an acceptance response is returned with a proposition of loan rate. Otherwise, a refusal response is returned. The assessor evaluates the risk of the loan. If the risk is low, an acceptance response is returned with a proposition of a loan rate. Otherwise (i.e., risk is high), the Approver is invoked.

Car Supplier: It has two essential methods: order and cancel.
order is invoked by a customer who specifies his need (e.g., type of car, maximum price, ...). The order is forwarded to two dealers (may be > 2 in the general case) by invoking Dealer1 and Dealer2. Each dealer returns a quote (price, delivery delay, ...). Car Supplier selects the dealer with the best quote (lowest price, or earliest delivery date if prices are equal) and sends him a definite offer which is also returned to the customer. When a dealer cannot satisfy the need, he answers with an infinite quote price.
cancel is invoked to abort an order before it is completed.

Car broker: It is built from Lender and Car Supplier. It is invoked by a customer who specifies his need (e.g., type of car, maximum price, ...). The need is forwarded by invoking the method order of Car Supplier. We have the following two situations:
If Car Supplier answers with an offer (because at least one of the dealers has given a finite quote price), Car Broker invokes the Lender who returns a refusal response or an acceptance with a loan rate.
If Car Supplier answers with an infinite quote price (because both dealers have given an infinite quote price), Car Broker replies with a refusal.
Every refusal (generated by Lender or Car Broker) is “caught” to cancel automatically the current order.

5.2. FI detection

Let us see how our FI detection method is applied to detect FI in the example of Section 5.1. Lender and Car Supplier are modeled in CRESS. Step 1 consists in composing the two WS to design Car Broker; this step is already done in [16]. In Step 2, we define the following variables, function and invariants:

Variables: price1, price2 are the quote prices offered by the two dealers;
price is the quote price offered by the selected dealer;
delivery1, delivery2 are the quote delivery delays offered by the two dealers;
delivery is the quote delivery delay offered by the selected dealer.

Function: Min(x,y) is the smallest of x and y.

Insets: (price1 ≠ price2) ⇒ (price = Min(price1,price2)),
(price1 = price2) ⇒ (delivery = Min(delivery1,delivery2)).

The two invariants model the condition used to select the dealer: lowest price or earliest delivery date if prices are equal. In Step 3, the reachability tree of the CRESS description of Car Broker is constructed and the above invariants are evaluated in each node during its construction. FI-nodes are detected.
After diagnosis, we realized that the FI is due to an error we unconsciously made when writing the condition to select a dealer in Car Supplier. The error is simply due to a move of parentheses. We have used the following condition to select dealer,

\[
\text{\textit{If ((price}_{1} \leq \text{price}_{2}) \text{ or (price}_{2} = \text{price}_{3}) \text{ and (delivery}_{1} < \text{delivery}_{2}))))
\]

6. Demonstration in the benchmark of [19]

Let us demonstrate our FI detection method in the examples of the benchmark of [19] which represents a fictitious virtual bookstore consisting of the following WS:

- **iPassport** is an identity management WS that simplifies authentication with multiple service providers.
- **PayMe** is a payment processing WS that allows payers to make secure payments online, and simplifies credit card processing for payees.
- **ShipEx** is a shipping WS that provides shippers with guaranteed delivery of product, and simplifies tracking of a shipment for shipees.
- **Shark** is a caching WS that improves performance by storing the results of previous requests.

Then, three composite WS **Amazin**, **Supplier** and **Customer** are constructed from the above individual WS. **Amazin** is a virtual bookstore which relies on a number of **Suppliers**, and gives **Customers** access to its virtual catalog and the option to order books from the catalog through an **Order Processing** feature.

For space limit, we do not discuss Step 4 of Section 4.4 in the following examples.

6.1. Example 1 of [19]: called “OrderProcessing – OrderProcessing”

The FI manifests itself by the existence of a blocking situation in the following way. Supplier, receives an order from a customer and forwards the order to Supplier because his stock is empty. Supplier in turn decides to forward the order to Supplier because its stock also is empty. Hence, we may reach the situation where each supplier is waiting the reception of the ordered book from the other supplier, which is clearly a blocking situation. Let us see how our FI detection method detects such FI.

Supplier and Supplier are modeled by CRESS diagrams Sup and Sup. In Step 1, Sup and Sup are composed into a single CRESS diagram Sup. In Step 2, the invariant “\(\text{NbEnabEvents} > 0\)” is selected, which means the absence of deadlock. In Step 3, the reachability tree of Sup is constructed and the invariant “\(\text{NbEnabEvents} > 0\)” is evaluated in each node during its construction. FI-nodes are detected.

6.2. Example 2 of [19]: called “Caching - Process Payment”

The FI manifests itself by the fact that, if an ordered book is in the cache (because it has been previously purchased), the process payment is court-circuited. Hence, the order is completed without payment. Let us see how our FI detection method detects such FI.

A supplier is modeled by a CRESS diagram Sup, where an order includes necessarily a process payment. And a caching service is modeled by a CRESS diagram Cache. In Step 1, Sup and Cache are composed into a single CRESS diagram SupCache. The composition is made by integrating Cache into Sup in the following way. When a book is ordered for the first time, the order is operated normally (including a process payment), and it is also stored in the cache. Then, when the book is subsequently ordered, the cached book is provided to the customer without process payment. In Step 2, the following two variables and invariant are defined:

Variables:
- **itemPaid** is a boolean variable specifying that the currently ordered book has been paid;
- **orderCompleted** is a boolean variable specifying that the current order is completed.
**Invariant:** \((\text{orderCompleted} = \text{true}) \Rightarrow (\text{itemPaid} = \text{true})\) is an invariant specifying that the order is completed only if the book has been paid.

In Step 3, the reachability tree of \(\text{SupCache}\) is constructed and the above invariant is evaluated in each node during its construction. FI-nodes are detected.

6.3. Example 3 of [19]: called “Order Processing – (Delivery or Process Payment)”

We consider two situations of FI that may occur when the order of a book is aborted (before its completion). These two FI are numbered 3.a and 3.b below:

3.a: *FI Called “Order Processing – Delivery” in [19]:* The FI manifests itself when, due to timing errors, a process payment is aborted while the delivery is completed. Hence, the possibility to receive a book which has not been paid (as in Example 2, but for a different reason).

3.b: *FI Called “Order Processing - Process Payment” in [19]:* The FI manifests itself when, due to timing errors, a delivery is aborted while the process payment is completed. Hence, the possibility to pay for a book which is not received.

A supplier is composed of several features such as: *Process Payment, Delivery, and Order Processing,* each one being described by a CRESS diagram. In Step 1, we obtain a CRESS diagram \(\text{Sup}\) modeling the supplier by composing the CRESS diagrams of those features. In Step 2, the following three variables and invariant are defined:

**Variables:**
- \(\text{orderAborted}\) is a boolean variable specifying that the current order is aborted;
- \(\text{itemPaid}\) (defined in Example 2);
- \(\text{itemDelivered}\) is a boolean variable specifying that the currently ordered item has been delivered.

**Invariant:**
- \((\text{orderAborted} = \text{true}) \Rightarrow (\text{itemPaid} = \text{false})\) and \((\text{itemDelivered} = \text{false})\) is an invariant specifying that the order is aborted only if the book has not been paid nor delivered.

In Step 3, the reachability tree of \(\text{Sup}\) is constructed and the above invariant is evaluated in each node during its construction. FI-nodes are detected; more precisely, nodes where \(\text{orderAborted} = \text{true}\) and \((\text{itemPaid} = \text{true})\) or \(\text{itemDelivered} = \text{true}\). When \(\text{orderAborted} = \text{true}\) and \(\text{itemPaid} = \text{false}\) and \(\text{itemDelivered} = \text{true}\), we are in the above case 3.a. When \(\text{orderAborted} = \text{true}\) and \(\text{itemPaid} = \text{true}\) and \(\text{itemDelivered} = \text{false}\), we are in the above case 3.b.

6.4. Example 4 of [19]: called “Order Processing - Fullfil Order”

As in Example 3, we consider a supplier which is composed of several features, each one being described by a CRESS diagram. The FI considered here is due to an ambiguity on the semantics of the price. More precisely, the FI manifests itself when some features use the term price, but assigning it different semantics. For example, one feature considers the price *including* taxes, while another feature considers the price *excluding* taxes.

In Step 1, as in Example 3, we have a CRESS diagram \(\text{Sup}\) modeling the supplier obtained by composing the CRESS diagrams of several features. Assume at least two features use the variable *price*. In Step 2, the variable *price* is renamed \(\text{price}_1\) and \(\text{price}_2\) in the two features, respectively, and the following semantic-invariant is defined: \(\text{price}_1 = \text{price}_2\). In Step 3, the reachability tree of \(\text{Sup}\) is constructed and the above invariant is evaluated in each node during its construction. FI-nodes are detected.
6.5. Examples 5, 6 and 7 of [19]: all of them associated to Access Profile

We consider Examples 5, 6 and 7 in this same section, because they correspond to several variants of the same problem: *non respecting the profile access policy*. Intuitively:

- In example 5 (called “Authenticate User - Access profile” in [19]): an untrusted supplier accesses some information in the profile of the customer.
- In example 6 (called “Access Profile - Access profile” in [19]): a trusted supplier accesses some information in the profile of the customer, which must be accessible uniquely to the customer.
- In example 7 (called “Manage Profile - Access profile” in [19]): a supplier accesses some information in the profile of the customer when the latter is not connected.

In Step 1, as in Examples 3 and 4, we have a CRESS diagram $Sup$ modeling the supplier obtained by composing the CRESS diagrams of several features. In Step 2, the following variables, functions and invariants are defined:

**Variables:**
- $owner$ = the owner of the profile;
- $isOnline = true/false$ means that the profile owner is/is not connected;
- $trustedAgents$ = set of agents trusted to access the non-confidential part of the profile.

**Functions:**
- $AccessedInfo(x)$ returns the information accessed by an agent $x$, (returns null when no information is accessed);
- $Confidential(info)$ returns a boolean value specifying whether the information info is confidential, i.e., must be accessed uniquely by the owner of the profile.

**Invariants:**
- $(Confidential(AccessedInfo(x)) = false) \Rightarrow (x \in trustedAgents) \text{ or } (x = owner))$
- $(Confidential(AccessedInfo(x)) = true) \Rightarrow (x = owner)$
- $(AccessedInfo(x) \neq null) \Rightarrow (x \in trustedAgents) \text{ or } (x = owner)$
- $(isOnline = false) \Rightarrow (trustedAgents = \emptyset)$
- $(isOnline = false) \Rightarrow (AccessedInfo(x) = null \ \forall \ agent \ x)$

The first invariant means that non-confidential information of the profile is accessible uniquely by the owner of the profile or a trusted agent. The second invariant means that confidential information of the profile is accessible uniquely by the owner of the profile. The third invariant means that information of the profile is accessible uniquely by the owner of the profile or a trusted agent. The fourth invariant means that no agent is trusted when the profile owner is not connected. The fifth invariant means that when the profile owner is not connected, his profile is inaccessible by the trusted agents.

In Step 3, the reachability tree of $Sup$ is constructed and the above invariants are evaluated in each node during its construction. FI-nodes are detected. Let us consider examples 5, 6 and 7:

**Example 5:** We have nodes where: $(supplier \neq owner) \text{ and } (supplier \notin trustedAgents) \text{ and } (AccessedInfo(supplier) \neq \emptyset)$. Hence, the above third invariant is not respected.

**Example 6:** We have nodes where: $(Confidential(AccessedInfo(supplier)) = true) \text{ and } (supplier \notin trustedAgents)$. Hence, the above second invariant is not satisfied.

**Example 7:** We have nodes where: $(isOnline = false) \text{ and } (AccessedInfo(supplier) \neq \emptyset)$. 
Hence, the above fifth invariant is not satisfied.

6.6. Example 8 of [19]: called “Order Processing - Order Processing”

The FI manifests itself by a blocking situation where Supplier$_1$ is waiting Supplier$_2$ who in turn is waiting Supplier$_3$, which corresponds exactly to Example 1. Hence Examples 1 and 8 can be considered as identical. But in Example 8, the FI is presented with a different viewpoint: None of the suppliers is available to the other one. Such FI can be detected in a different way than in Example 1, as follows. As in Example 1, Supplier$_1$ and Supplier$_2$ are modeled by CRESS diagrams Sup$_1$ and Sup$_2$. In Step 1, Sup$_1$ and Sup$_2$ are composed into a single CRESS diagram Sup. In Step 2, the following variables and invariant are defined:

**Variables:**
- available$_1$ is a boolean variable of Sup$_1$ specifying that Supplier$_1$ is available, hence not blocked.
- available$_2$ is a boolean variable of Sup$_2$ specifying that Supplier$_2$ is available, hence not blocked.

**Invariant:** (available$_1$ = true) and (available$_2$ = true).

The invariant means that both suppliers must be available in any situation. In Step 3, the reachability tree of Sup is constructed and the above invariant is evaluated in each node during its construction. FI-nodes are detected; more precisely, nodes where (available$_1$ = false) and (available$_2$ = false).

7. Conclusion

We have developed a generic off-line method to detect FI in WS during their design (from scratch or by composing existing Web services). The method uses model-checking techniques to detect FI which are specified by formal expressions called invariants. The two keywords off-line and generic are related to two of our contributions. For the off-line aspect, we justify that the off-line is mandatory and the on-line approach is unrealistic. By generic, we mean that our method is applicable for the detection of any FI that can be specified by invariants.

As a future work, we plan to develop an off-line FI resolution method, which resolves the detected FI. Another planned future work is to develop a prototype to analyze and validate our FI detection and resolution methods.

**Acknowledgements:** I would like to thank Dr. Zohair Chentouf for his comments that contributed in improving this document.

**REFERENCES**


