A Use-Modify Framework to Detect Feature Interactions in Web Services

Ahmed Khoumsi and Zohair Chentouf*

Department of Elect. & Comp. Eng., University of Sherbrooke, Sherbrooke, Canada, and *Software Engineering Department, College of Information and Computer Sciences, King Saud University, Riyadh, KSA

zchentouf@ksu.edu.sa

Abstract. Composing Web services is often beneficial since created the new Web services from existing ones. However, Web service composition is prone to feature interactions, which denote undesirable behaviors arising when several Web services are used together. The existing methods for detecting feature interactions suffer generally from state space explosion. In this paper, we develop a method to detect feature interactions in Web services, which targets the reduction of state space explosion while trying to keep an acceptable power of feature interaction detection. The proposed method is based on the use of a language called Use-Modify which models Web services at a high abstraction level. A Use-Modify model of a Web service provides information such as “who uses what”, “who modifies what”, and characterizes each operation of use and modifying by “always”, “sometimes”, “never” and “maybe”. "Use-Modify" also indicates, for each use and modifies, whether there are conditions which may specified or unspecified. We study the computational complexity of our feature interaction detection method and demonstrate its applicability in several examples.

Keywords: Composing web services; Feature interaction detection; High abstraction level; Use-Modify relation; Use-Modify model.

1. Introduction

When existing Web Services (WS) are composed to create new WSs, the latter can contain undesired behaviors, which are called feature interactions (FI). Here is an example of FI in WS: we consider a supplier to which orders can be sent. When his stock is empty, a supplier forwards
any incoming order to another supplier. Consider two WSs Supplier\(_1\) and Supplier\(_2\), an assuming that an order is sent to Supplier\(_1\) and those both Suppliers have their stocks empty. We may have a following situation: Supplier\(_1\) forwards an order to Supplier\(_2\) which in turn the forwarding of the order to Supplier\(_1\). The FI manifests itself by a blocking situation where each supplier is waiting the answer of the other.

FIs have been intensively studied in telecommunication services (or Telecom-services)\(^{[1-9]}\), and ever since more recently in WSs. Many methods have been developed to detect FIs, some of them are rigorous and have a high power of FI detection. But the latter suffer from state space of explosion, such as those applying model-checking techniques. The approach we are proposing to detect FIs in WSs targets, the reduction of such a state space explosion problem while trying to keep an acceptable power of FI detection. We model the behaviors of WSs by so-called Use-Modify language (or UM-language) which is a high abstraction level formalism whose basic principle is to specify “who uses what” and “who modifies what”. UM-language permits also to characterize each “use” and “modify” by “always”, “sometimes”, “never” or “maybe”. Moreover, UM-language may also indicate conditions to “use” or “modify”.

Our Use-Modify approach is slightly inspired by many workers\(^{[10,11]}\). Our contribution is that while\(^{[10-11]}\) are mainly based on intuitive ideas, we adopt a much more rigorous approach where all our ideas are studied thoroughly and formally. A much shorter version of our paper is published in 2012\(^{[12]}\).

The structure of the paper and its contributions compared to Khoumsi, et al.,\(^{[12]}\) are as follows:

- In Section 2, we explain some fundamental differences between composing WSs and composing Telecom-services.
- Section 3 presents some related work on modeling and composing WSs and detecting their FIs.
- In Section 4, we propose a Use-Modify language (or UM-language) to model WSs at a high abstraction level. A UM-model is a set of UM-relations like “L uses R” or “L modifies R”, where L and R represent WSs, functionalities of WSs or variables of WSs, and each “use” and “modify” is characterized by “always”, “sometimes”, “never” or “maybe”. Here are specific contributions in comparison with those of Khoumsi, et al.,\(^{[12]}\).
- the semantics of “use” and “modify” and their characterizations are defined more clearly and rigorously (Sect. 4.2);
- The identified Formal conditions to characterize UM-relations as well-formed (Sect. 4.3);
- Formal conditions can be associated with UM-relations to restrict their general semantics (Sect. 4.5).
- Section 5 proposes a number of logical rules that permit to enrich a UM-model in function of deriving new UM-relations from given UM-relations. Here are specific contributions in comparison with\cite{12}:
  - We first define fundamental rules which do not refer to UM-relations (Section 5.1); they are rather based on general logical statements; these rules are absent in Khoumsi, et al.,\cite{12}.
  - Three categories of UM-rules (i.e. applicable rules of UM-relations) are deduced from the fundamental rules. (Section 5.2-5.4), instead of being partially defined without justification and categorization (as in Khoumsi, et al.,\cite{12}).
  - Soundness and completeness of the fundamental rules and the UM-rules are rigorously studied and discussed (Sections 5.5-5.6).
  - Utility of characterizing “use” and “modify” by “maybe” is explained (Section 5.6.3).
- In Section 6, we are going to present a Use-Modify-based & method of detecting FIs in WSs. Contrary to that of Khoumsi, et al.,\cite{12}:
  - The method is specified as: Three-steps first algorithm where we indicate clearly what is done automatically and what is done by the designer.
  - Second (which is not an easily understandable\cite{12}) is illustrated by an abstract example throughout Sect. 6.2.
  - We study the computational complexity of our FI detection method.
- Section 7 to demonstrates the applicability of our method for detecting all of the FIs of the benchmark\cite{13} is a part of the FI\cite{14}.
- In Section 8, we demonstrate that our method can be used to detect several FIs, and we never only consider the WS composition, but also a Telecom-service composition and a mixed composition of WS including to the Telecom-service\cite{15}.
- Section 9 conclusions and by recapitulating of the contributions and proposing are some of the future work.
- Section 10 contains the proofs of some propositions; where Khoumsi, et al.,\textsuperscript{[12]} not all of the propositions.

2. Web Service Composition Versus Telecommunication Service Composition

Let us show that composing WSs is different from composing Telecom-services.

1. Telecom-services can generally be abstracted by a few parameters. For example\textsuperscript{[16]}, each service is abstracted by a triggering party, and origin and destination parties. The services\textsuperscript{[17]}, are abstracted by some processing points that correspond to the main steps in a phone call. On the other hand contrary, WSs cannot be so simply abstracted, because a WS can be provided any imaginable software system providing a service through the Web.

2. Composing two Telecom-services generally means running them in parallel. Most of the FI studies for Telecom-services are based on this simple composition approach. On the contrary, WS composition means designing a new WS by composing existing WSs, based on the principle of software reusability. Hence, WS composition requires a design phase.

We deduce that WS composition may be much more complex than composing Telecom-services, so that cannot be automated in general. To address the complexity of WS composition, in several models, those have been developed, such as orchestration and choreography.

Now, let us draw the attention of the reader to an important difference between Telecom-services and WSs in FI detection. The presence of FIs between the two composed Telecom-services depend generally unique on those composed services, because the composition consists simply in running the services in parallel. On the contrary, the presence of FIs in WSs depends generally on the way of the WSs have been composed, because there are many ways to compose WSs.
3. Related Work on Modeling and Composing WSs and Detecting Their FIs

An important contribution\cite{13, 14} is to raise the interest of researchers to the problem of WS composition and FI detection\cite{13}. That presents a case of study which can be used as a benchmark to assess FI detection methods. Another contribution in raising an interest can be found in\cite{15}, which shows that FIs of Telecom-Services are different from FIs in WSs.

Let the term on-line (resp. off-line) qualify the methods which are applicable during in the execution (resp. design). On-line WS composition and FI detection methods are studied for example in\cite{18-20}. \cite{18} presents an on-line of FI detection method inspired from the Situation of Calculus. \cite{19} presents an on-the-fly approach to compose WSs. \cite{20} identify some challenges and opportunities in on-line FI detection and resolution.

Much more work has been done in off-line WS composition and FI management, e.g. in\cite{21-25}. \cite{21} proposes an off-line FI detection method using Label Transitions Systems (LTS). \cite{22} proposes a method based on Petri nets that detects one type of FIs: race conditions. \cite{23} uses Petri nets to describe WSs and presents simple examples for merging WS descriptions. \cite{24} presents an FI detection method using the model-checker UPPAAL; WSs are described in WS-BPEL which is translated into timed automata. \cite{25} presents an FI detection method that uses the model-checker SPIN\cite{26}; WSs are described in BPEL4WS\cite{27} which is translated into Promela.

Some work on user-interfacing and software-tooling for WS composition can be found in\cite{28, 29}. \cite{28} proposes an environment using Mashup for WS composition, and\cite{29} presents an integrated development environment for WS composition. FI detection is not studied in\cite{28, 29}.

\cite{30, 31} propose an extension of the business model of\cite{32} to support WS composition. The authors of\cite{30, 31} go further in\cite{33} by studying how WSs can be categorized and assembled. FI detection is not studied in\cite{30, 31, 33}.

\cite{34, 35} contain a rigorous study of WS composition, where theoretical, software-tooling and user-interfacing aspects are considered. The CRESS formalism is used which can be automatically translated into BPEL and LOTOS.
4. Use-Modify Language to Model WSs

In the references of Section 3 that study FI detection, the developed FI detection methods may suffer from state space explosion, because they are based on formalisms specifying WS behaviors exhaustively. The approach we adopt targets to avoid state space explosion while keeping an acceptable power of FI detection. For that purpose, we develop a so-called Use-Modify language (or UM-language) to model WSs at a high abstraction level, whose principle is to specify “who uses what” and “who modifies what”. Such an omission of details is motivated by the desire to avoid state space explosion during FI detection. With Use-Modify, WSs are specified at two levels: their interfaces are specified like objects in object-oriented analysis (OOA); and their behaviors are specified by what called is Use-Modify relations (UM-relations) in the form of “L uses of Y” or “L modifies to R”. L and R correspond to WSs, functionalities of WSs or variables of WSs, and either “use” and “modification” are characterized by “always”, “sometimes”, “never” or “maybe”. A set of UM-relations modeling the behavior of the WS is called its behavior model, or its UM-model to emphasize the use of UM-relations. The UM-model describes a WS logically, in the sense that it specifies how a WS behaves but it does not necessarily correspond to its implementation. The UM-model is targeted uniquely to be manipulated by our proposed FI detection method which will be presented in Section 6. While designing (and pre deploying) a WS, a UM-model of such a WS must be constructed and analyzed to determine whether the WS is FI prone. Therefore, our method is off-line.

4.1. Interface Model Based on Object-Oriented Analysis

The interface of a WS is modeled as a class skeleton in OOA, and the interface of each executable instance of WS is modeled as an object skeleton of a class. By skeleton, we mean that the classes and objects are specified by attributes and methods signatures. A method signature specifies a function by its name, its input and/or output parameters and its returned result (if any), and without a body. Object skeleton corresponds to interface in Java. Hence, the behavior is not specified. For the sake of brevity, we will omit the terms skeleton and signature in class skeleton, object skeleton, and method signature. There exist two types of attributes: basic attributes and complex attributes. Basic attributes are
variables of primitive types, like int, float, double, boolean. Complex attributes are objects. For the sake of clarity, methods, basic attributes and complex attributes are named differently as follows:

- Basic attributes (or primitive variables): they are named in italic with the first letter non capitalized. For example, risk, rate, and amount.
- Complex attributes (or objects): they are named in italic with the first letter capitalized. For example, Assessor, Approver, Lender, Supplier.
- Methods: they are named in italic with the first letter non capitalized, and they terminate by (). For example, quote(), approve() and assess().

As in OOA, attribute a and a method m() of an object O are referred to as O.a and O.m(), respectively. The object name O can be omitted when there is no ambiguity or when it is irrelevant. We will use the notions of feature and WS as follows:

- Feature: it is a basic WS which is not composed of other WSs. A feature is modeled as an object whose all attributes are basic. When several similar features are used, the latter can be modeled as objects of the same class. A class is named with all letters capitalized, for example, SUPPLIER.
- WS: it is a complex WS created by composing features and/or WSs. Like features, WSs can be modeled by objects and classes. The fact that a WS is composed of several objects (WSs and/or features) implies that it has a complex attributes.

Let us consider some examples of features and WSs taken from [35] and give an idea of how they can be modeled as objects. We do not present them in detail, we just indicate one or two attributes and methods for each feature or WS.

**Example 1:** The feature Approver has a method approve() and two basic attributes amount and rate. approve() evaluates a loan of a given amount and refuses or approves it. A rate is selected if the loan is approved.

**Example 2:** The feature Assessor has a method assess() and three basic attributes amount, risk and rate. assess() evaluates the risk of a loan of a given amount. If risk is low, an acceptance response is returned with a proposed loan rate, otherwise a refusal is returned.
Example 3: The WS Lender is composed of the two features Approver and Assessor. Lender has two attributes that correspond to Approver and Assessor. Lender has also a method quote() and a basic attribute amount. The method quote() approves or assesses a loan of a given amount in the following way: quote() invokes the method approve() of Approver if amount $\geq$ 10000, or the method assess() of Assessor if amount $< 10000$. quote() also invokes approve() if assess() returns a refusal.

We have shown how WSs have their interfaces (and not their behaviors) modeled as classes and objects. Note that these interfaces can be visualized as a subset of UML class diagrams where the unique associations are compositions and aggregations, which may seem too restrictive compared to UML class diagrams. This restriction is justified by the fact that our interfaces will be used uniquely to detect FIs at a high abstraction level. These interfaces do not reflect necessarily the implementation structures of WSs, while UML class diagrams can be used to model implementations, and hence may need to be closely associated to implementations structures.

Interfaces do not give any information on how WSs behave. In the above three examples, the behaviors were indicated for information, they are not described in the objects. In the remainder of Section 4, we show how WSs have their behaviors modeled at a high abstraction level by the Use-Modify formalism.

4.2. Introduction to the Use-Modify Formalism

A method is said active if its execution modifies (sometimes or always) the value of some attribute (of any object). An object is said active if it contains an active method or a complex attribute which is an active object. A basic attribute cannot be active. A method or object is said passive if it is not active. Intuitively, an active object is an object that permits to modify some attribute (of any object). Let active access to an attribute mean an access that modifies the attribute. Hence, we categorize accesses in two actions: “use” and “modify” which will be characterized by various “intensities”. Let us first consider the “use” of the action:

- “use!” means “has always access to”.
- “use?” means “has sometimes access to”; by sometimes, we mean under some specified or unspecified conditions which happen to be true (i.e. the conditions cannot be always false).
- “use%” means “has never access to”.
- “use#” means “has maybe access to”, i.e., we do not know if there is an access.

In the same way, the action “modify” is used with various “intensities” as “modify!”, “modify?”, “modify%” and “modify#”. The difference between “use” and “modify” is that “modify” corresponds to an active access, while “use” corresponds to an access which may be passive or active.

To clarify particularly the semantics of “always”, “sometimes”, “maybe” and “never”, we detail below the different types of so-called Use-Modify relations (or UM-relations):

- “L use! R” means that R is accessed each time and L is applied.
- “L use? R” means that R is accessed in some (known or unknown) situation(s) where L is applied. Note that this case may include the following two cases:
  - L has access to R in some situations not in all situations;
  - L has access to R in all situations.
- “L use% R” means that L never uses R.
- “L use# R” means that we suspect that L uses R, but we are not certain.

- “L modify! R” strengthens “L use! R” by specifying that the access is active, i.e. R is modified each time L is applied.
- “L modify? R” strengthens “L use? R” by specifying that the access is active, i.e. R is modified in some (known or unknown) situation(s) where L is applied. Note that this case may include the following two cases:
  - L modifies R in some situations not in all situations;
  - L modifies R in all situations.
- “L modify% R” means that L never modifies R.
- “L modify# R” means that we suspect that L modifies R, but we are not certain.

Note that use# is less precise than use!, use? and use%, and modify# is less precise (we also say: weaker) than modify!, modify! and modify%. use# and modify# have been defined and we will show that if they can be deduced by some rules. Typically, a UM-relation “L use# R” is irrelevant (hence of that should be removed) so if we have one of its stronger UM-
relations “L use! R”, “L use? R” or “L use% R”. In the same way, a UM-relation “L modify# R” is irrelevant (so that should be removed) if we have one of its stronger UM-relations “L modify! R”, “L modify? R” or “L modify% R”. We will return to this aspect in Section 5.6.3.

In the sequel, “!”,”?”,”%” and “#” are not written in some contexts where they are irrelevant. In this case, we write “use” to mean “use!”, “use?”, “use%” or “use#”, and we write “modify” to mean “modify!” or “modify?”,”modify%” or “modify#”.

4.3. Well-formed UM-relations “L use R” and “L modify R”

In this subsection, we still clarify more the semantic of UM-relations “L use R” and “L modify R” and we present restrictions on R and L that are necessary and sufficient to characterize a UM-relation as well-formed.

4.3.1. UM-relation “L use R”

In a UM-relation “L use R”:
- R can be a method \( m() \): “L use \( m() \)” means that L calls \( m() \);
- R can be a basic attribute \( x \): “L use \( x \)” means that L reads or changes the value of \( x \).
- R can be a complex attribute, i.e. R is an object which may have its own (basic and complex) attributes and/or methods: “L use R” means that L uses one or more of the attributes or methods of R.

In the above three cases, we have the actions “calls”, “reads or changes” and “uses”, respectively. We refer to any of these actions by “action on R”. The 3 cases are generic since we have “use” without !, ?, # or %. Let us see what we obtain if we replace the generic “use” by use!, use?, use# or use? :
- With use!: we have to characterize the action on R by “always”,
- With use?: we have to characterize the action on R by “sometimes”,
- With use#: we have to characterize the action on R by “maybe”,
- With use%: we have to characterize the action on R by “never”.

Let us now see the conditions on L in a UM-relation “L use R”:
- L can be a method \( p() \): the action on R is realized by the execution of \( p() \).
- L can be a complex attribute: there are two possible situations:
  - L has a method that realizes the action on R;
  - L has a complex attribute that realizes the action on R.
- L cannot be a basic attribute: indeed, a basic attribute can uniquely be read and modified.

4.3.2. **UM-relation “L modify R”**

A difference with “L use R” is that in “L modify R”, R cannot be a method, because it is a nonsense to modify a method. The latter can uniquely be called (i.e. used). Hence, in a UM-relation “L modify R”:
- R cannot be a method \( m() \): a method can only be used (by calling it);
- R can be a basic attribute \( x \): “L modify \( x \)” means that L changes the value of \( x \).
- R can be a complex attribute, i.e. R is an object which may have its own (basic and complex) attributes and/or methods:
  - “L modify R” means that L modifies one or more of the attributes or methods of R.

In the above two “can be” cases, we have the actions “changes” and “modifies”, respectively. We refer to any of these actions by “active action on R”. The 2 cases are generic since we have “modify” without !, ?, # or %. Let us see what we obtain if we replace the generic “modify” by modify!, modify?, modify# or modify? :
- With modify! : we have to characterize the active action on R by “always”,
- With modify? : we have to characterize the active action on R by “sometimes”,
- With modify#: we have to characterize the active action on R by “maybe”,
- With modify%: we have to characterize the active action on R by “never”.

The conditions on L in a UM-relation “L modify R” are the same conditions identified for “L use R” in Subsection 4.3.1.

**Definition 4.1 (Well-formed UM-relation)** A UM-relation “L use R” (resp. “L modify R”) is said well-formed if it respects the conditions of Subsection 4.3.1 (resp. 4.3.2).
4.4. Examples of UM-models

**Example 4:** Here are some UM-relations that can be derived from the literal descriptions in Examples 1, 2, 3 of Section 4.1:

**Approver** (of example 1):
M1: Approver.approve() modify! Approve.amount // approve()
   // sets amount by a value received as input argument
M2: Approver.approve() modify? Approver.rate //approve() computes
   //rate if loan accepted

**Assessor** (of example 2):
M3: Assessor.assess() modify! Assessor.amount // assess() sets amount
    // by a value received as input argument
M4: Assessor.assess() modify! Assessor.risk // assess() computes the
    //risk
M5: Assessor.assess() modify? Assessor.rate //assess() computes the
    //rate if the risk is low

**Lender** (of example 3): Since Lender is composed of Approver and Assessor, its model contains the UM-relations M1-M5. Additional UM-relations are necessary to model the coordination of Approver and Assessor by Lender. Here are examples of such additional UM-relations:
M6: Lender use! Lender.quote() // Lender starts by the execution of
    //its method quote()
M7: Lender.quote() modify! Lender.amount // quote() sets amount by a
    //value received as input argument
M8: Lender.quote() use? Approver.approve() // quote() calls approve()
    // if amount ≥ 10000 or if assess() refuses the loan
M9: Lender.quote() use? Assessor.assess() // quote() calls assess() if
    // amount < 10000

**Example 5:** Let us use the benchmark of [13] to present other examples of use? and modify?. Examples 5, 6 and 7 of this benchmark are related to accessing the user profile. We consider a WS Supplier that needs to have access to user profiles. We assume that each profile contains two parts: a confidential part and a public part. The two parts can be read and modified by the profile of the owner. The confidential part can also be read by some trusted entities, while the public part can be read by anyone.

All what concerns a user is represented as an object User with an attribute profile. The latter represents the user profile which is itself an
object with two attributes \textit{conf} and \textit{pub}, for the confidential and public parts respectively. Here are some UM-relations where \textit{Supplier} is a trusted or untrusted supplier.

N1: \textit{Supplier} use? \textit{User}.\textit{profile} // \textit{Supplier} can read \textit{profile} with the // following restriction: \textit{Supplier} can read the confidential // part only if he is trusted.

N2: \textit{Supplier} modify\% \textit{User}.\textit{profile} // \textit{Supplier} cannot modify // \textit{profile}

N3: \textit{Supplier} use? \textit{User}.\textit{profile}.\textit{conf} // \textit{Supplier} can read \textit{conf} // only if he is trusted

N4: \textit{Supplier} modify\% \textit{User}.\textit{profile}.\textit{conf} // \textit{Supplier} cannot modify // \textit{conf}

N5: \textit{Supplier} modify\% \textit{User}.\textit{profile}.\textit{pub} // \textit{Supplier} cannot modify // \textit{pub}

4.5. 	extbf{Conditions Associated to UM-Relations}

In a UM-relation “L x R”, we may specify conditions as follows:

L x R : [condition1, condition2, …]

Consider for example a WS \textit{Supplier} to which an order can be sent, e.g., by calling its method \textit{order()}. \textit{Supplier} can itself call the \textit{order()} method of another supplier of the same class \textit{SUPPLIER}. This is specified by the UM-relation “\textit{Supplier}.\textit{order()} use? \textit{SUPPLIER}.\textit{order()}”. Assuming a supplier does not call its own \textit{order()} method, we associate to this UM-relation a condition stating that \textit{SUPPLIER} does not comprise \textit{Supplier}. Formally:

\textit{Supplier}.\textit{order()} use? \textit{SUPPLIER}.\textit{order()} : [\textit{SUPPLIER} \neq \textit{Supplier}].

This condition will be reconsidered in the example of Section 7.1.

Conditions can also be useful in a UM-relation with “use?” or “modify?” to justify why we have not “use!” or “modify!” in the considered UM-relation. Consider for example a supplier who accesses some information in the profile of a customer only if he is authorized. This can be modeled as follows:

\textit{Supplier} use? \textit{profile} : [\textit{Supplier}.\textit{authorized} = true].
This kind of condition will be used to define a FI pattern, namely Pattern 4 of Section 6.3. It will be illustrated by an example in Section 7.5.

5. Logical Rules of Use-Modify Language

To make UM-modeling applicable in a rigorous way, we provide in this section a set of logical rules that can be used in the phase of construction of UM-relations modeling a WS or several interacting WSs. We will consider three types of rules:
- **implication UM-rules**: they permit to deduce a new UM-relation from an existing UM-relation;
- **fusion UM-rules**: they permit to deduce a new UM-relation from two existing UM-relations;
- **contradiction UM-rules**: they permit to identify incompatible UM-relations.

Let us first give in Section 5.1 fundamental rules from which the three types of UM-rules will be synthesized in Sections 5.2-5.4. By fundamental, we mean that the rules of Section 5.1 are based on general logical statements; they do not refer directly to UM-relations. Sections 5.5-5.6 are related to soundness and completeness of the fundamental and UM-rules. Section 5.7 illustrates the use of UM-rules.

5.1. Fundamental rules

The objective of this subsection is to identify a set of fundamental rules that specify:
- links between “use” and “modify” (R1, R2);
- links between “always”, “sometimes” and “never” (R3-R5);
- How “use” can be combined with other actions by transitivity (R6-R9).

Note that these rules are not specified formally because their objective is to present fundamental principles which will justify the formal rules of Sections 5.2-5.4.

5.1.1. Links between “use” and “modify”

The action “use” refers to any active or passive access. That is, “L uses R” means that L has an access to R which may or may not modify the state of R. The action “modify” is an active “use”, *i.e.* “L modifies R”
means that L has a particular use of R that modifies its state. Hence, L can modify R only by using it, or in other terms, L cannot modify R if L does not use R. Therefore we have the following two rules R₁ and R₂ which are in fact equivalent:

- **R₁**: “L modifies R” implies “L uses R”;
- **R₂**: “L does not use R” implies “L does not modify R”.

### 5.1.2. Links between “always”, “sometimes” and “never”

In Section 4.2, we have explained our exact semantics of “always”, “sometimes”, “never” and “maybe”, from which the following rules R₃-R₅ can be easily understood. Note that “maybe” does not intervene in these rules; this is because our semantics of “maybe” is too coarse and corresponds to a “don’t know” situation.

- **R₃**: “L always makes an action A” implies “L sometimes makes A”;
- **R₄**: “L never makes an action A” and “L sometimes makes A” are contradictory;
- **R₅**: “L never makes an action A” and “L always makes A” are contradictory.

### 5.1.3. Combining “use” with other actions by transitivity

Consider actors U, L and R, such that U always applies an action A to R. Our semantics of “always” (Section 4.2) means that each time U is used, it inevitably applies the action A to R. Consider the following two cases:

- Assume that L sometimes uses U, *i.e.* there is at least one case where L uses U. Hence, we deduce logically that there is at least one case where L applies the action A to R, *i.e.* L sometimes applies the action A to R. This leads to rule R₆ below.

- Assume that L always uses U, *i.e.* each time L is used, it uses U. Hence, we deduce logically that each time L is used it applies the action A to R, *i.e.*, L always applies the action A to R. This leads to rule R₇ below.

Assuming that U always applies the action A to R:

- **R₆**: “L sometimes uses U” implies “L sometimes applies A to R”;
- **R₇**: “L always uses U” implies “L always applies A to R”.

Consider now actors U, R and L, such that U sometimes applies an action A to R. Our semantics of “sometimes” (Section 4.2) means that there is at least one case where U applies the action A to R. Consider the following two cases:

- Assume that L sometimes uses U, \textit{i.e.} there is at least one case where L uses U. We cannot deduce anything about the application of A by L, for the following reason: the cases where U applies A to R are not necessarily the cases where L uses U. Hence, we can only deduce that L maybe applies action A to R, which corresponds to rules R8.

- Assume that L always uses U, \textit{i.e.} each time L is used, L uses U. We cannot deduce anything about the application of A by L, for the following reason: the cases where U applies A to R are not necessarily the cases where L is used. Hence, we can only deduce that L maybe applies action A to R, which corresponds to rules R9.

Assuming that U sometimes applies the action A to R:

- \textbf{R8}: “L sometimes uses U” implies “L maybe applies A to R”;
- \textbf{R9}: “L always uses U” implies “L maybe applies A to R”.

5.1.4. Recapitulation of the fundamental rules R1-R9

\textbf{R1}: “L modifies R” implies “L uses R”.
\textbf{R2}: “L does not use R” implies “L does not modify R”.
\textbf{R3}: “L always makes an action A” implies “L sometimes makes A”.
\textbf{R4}: “L never makes an action A” contradicts “L sometimes makes A”.
\textbf{R5}: “L never makes an action A” contradicts “L always makes A”.

Assuming that U always applies an action A to R:

- \textbf{R6}: ”L sometimes uses U” implies “L sometimes applies A to R”;
- \textbf{R7}: “L always uses U” implies “L always applies A to R”.

Assuming that U sometimes applies an action A to R:

- \textbf{R8}: “L sometimes uses U” implies “L maybe applies A to R”;
- \textbf{R9}: “L always uses U” implies “L maybe applies A to R”.

From R1-R9, we define in Sections 5.2-5.4 three types of specific UM-rules (\textit{i.e.} rules on UM-relations): implication UM-rules, fusion UM-rules, and contradiction UM-rules. These UM-rules are identified in the form I_n, F_n and C_n, respectively, and also in a mnemonic form R[…], that may help to guess the statement of each rule.
5.2. Implication UM-Rules

In this subsection, we present implication UM-rules, i.e. we identify cases where a UM-relation implies another UM-relation. The implication UM-rules I₁-I₅ below are deduced from Rules R₁-R₃ of Section 5.1. More precisely:

- I₁ and I₂ are the translations of R₁ into UM-rules by characterizing the actions by “always” and “sometimes”, respectively.
- I₃ and I₄ are the translations of R₃ into UM-rules by using the actions “modify” and “use”, respectively.
- I₅ is the translation of R₂ into UM-rule.

The condition associated to I₅ is required to guarantee that the derived “L modify%” is well-formed assuming that “L use%” is well-formed. This condition is necessary because the “well-formed” constraints of “L use% R” (in Subsection 4.3.1) are weaker than the “well-formed” constraints of “L modify%” (in Subsection 4.3.2). The UM-rules I₃-I₄ do not require conditions because the “well-formed” constraints of their left members are the same as the “well-formed” constraints of their right members. The UM-rules I₁-I₂ do not require conditions because the “well-formed” constraints of their left members are stronger than the “well-formed” constraints of their right members.

\[
\begin{align*}
I₁ & : \text{R}[m\text{!!}=>u\text{!!}]: \quad \text{“L modify! R” } \Rightarrow \quad \text{“L use! R”} \\
I₂ & : \text{R}[m?=>u?]: \quad \text{“L modify? R” } \Rightarrow \quad \text{“L use? R”} \\
I₃ & : \text{R}[m\text{!!}=>m\text{?}]: \quad \text{“L modify! R” } \Rightarrow \quad \text{“L modify? R”} \\
I₄ & : \text{R}[u\text{!!}=>u?]: \quad \text{“L use! R” } \Rightarrow \quad \text{“L use? R”} \\
I₅ & : \text{R}[u\text{%}=>m\%]: \quad \text{“L use% R” } \Rightarrow \quad \text{“L modify% R” if}
\end{align*}
\]

Assuming that the conditions of Section 4.3.2 are respected by L and R:

\[
\begin{align*}
I₅ & : \text{R}[u\text{%}=>m\%]: \quad \text{“L use% R” } \Rightarrow \quad \text{“L modify% R” if}
\end{align*}
\]

5.3. Fusion UM-Rules

In this subsection, we present fusion UM-rules, i.e. we identify cases where two UM-relations derive another UM-relation. The fusion rules F₁-F₄ below are deduced from Rules R₆-R₇ of Section 5.1.3 as follows:

- F₁ is the translation of R₇ into UM-rule by taking action A as “use R”,
- F₂ is the translation of R₆ into UM-rule by taking action A as “use R”,
Ahmed Khoumsi and Zohair Chentouf

- F₃ is the translation of R₇ into UM-rule by taking action A as “modify R”.  
- F₄ is the translation of R₆ into UM-rule by taking action A as “modify R”.  

The UM-rules F₅-F₈ below are deduced by combining I₁-I₂ and F₁-F₄ as follows:

- F₅ is deduced from I₁ and F₁,  
- F₆ is deduced from I₂ and F₂,  
- F₇ is deduced from I₁ and F₃,  
- F₈ is deduced from I₂ and F₄.

F₁: R[u!u!=>u!]: “L use! U” and “U use! R” => “L use! R”  
F₃: R[u!m!=>m!]: “L use! U” and “U modify! R” => “L modify! R”  
F₄: R[u?m!=>m?]: “L use? U” and “U modify! R” => “L modify? R”  
F₅: R[m!u!=>u!]: “L modify! U” and “U use! R” => “L use! R”  
F₆: R[m?u!=>u?]: “L modify? U” and “U use! R” => “L use? R”  
F₇: R[m!m!=>m!]: “L modify! U” and “U modify! R” => “L modify! R”  
F₈: R[m?m!=>m?): “L modify? U” and “U modify! R” => “L modify? R”

The UM-rules F₉-F₁₂ below are deduced from R₈-R₉ of Section 5.1.3 as follows:

- F₉ is the translation of R₉ into UM-rule by taking action A as “use R”,  
- F₁₀ is the translation of R₈ into UM-rule by taking action A as “use R”,  
- F₁₁ is the translation of R₉ into UM-rule by taking action A as “modify R”,  
- F₁₂ is the translation of R₈ into UM-rule by taking action A as “modify R”.  

Note that F9 and F11 can also be deduced as follows:
- F9 is deduced from I4 and F10,
- F11 is deduced from I4 and F12.
We have also the UM-rules F13-F16 which can be deduced as follows:
- F13 is deduced from I1 and F9,
- F14 is deduced from I2 and F10,
- F15 is deduced from I1 and F11,
- F16 is deduced from I2 and F12.
Note that F13 and F15 can also be deduced as follows:
- F13 is deduced from I3 and F14,
- F15 is deduced from I3 and F16.

F9: $R[u!u?=u\#]$:

\[
\text{“L use! U” and “U use? R” } \Rightarrow \text{“L use\# R”}
\]

F10: $R[u?u?=u\#]$

\[
\text{“L use? U” and “U use? R” } \Rightarrow \text{“L use\# R”}
\]

F11: $R[u!m?=m\#]$

\[
\text{“L use! U” and “U modify? R” } \Rightarrow \text{“L modify\# R”}
\]

F12: $R[u?m?=m\#]$

\[
\text{“L use? U” and “U modify? R” } \Rightarrow \text{“L modify\# R”}
\]

F13: $R[m!u?=u\#]$

\[
\text{“L modify! U” and “U use? R” } \Rightarrow \text{“L use\# R”}
\]

F14: $R[m?u?=u\#]$

\[
\text{“L modify? U” and “U use? R” } \Rightarrow \text{“L use\# R”}
\]

F15: $R[m!m?=m\#]$

\[
\text{“L modify! U” and “U modify? R” } \Rightarrow \text{“L modify\# R”}
\]

F16: $R[m?m?=m\#]$

\[
\text{“L modify? U” and “U modify? R” } \Rightarrow \text{“L modify\# R”}
\]

5.4. Contradiction UM-Rules

In this subsection, we present contradiction UM-rules, i.e. we identify pairs of UM-relations which are incompatible (or mutually exclusive) with each other. A UM-model containing pairs of incompatible UM-relations is inconsistent and hence may be a symptom of FI. The four contradiction UM-rules C1-C4 below are deduced from Rule R4-R5 of Section 5.1.2 as follows:
- C1 is the translation of R4 into UM-rule, by taking action A as “modify R”,

- C2 is the translation of R4 into UM-rule, by taking action A as “use R”,
- C3 is the translation of R5 into UM-rule, by taking action A as “modify R”,
- C4 is the translation of R5 into UM-rule, by taking action A as “use R”.

Note that C3 and C4 can also be deduced as follows:
- C3 is implied from I3 and C1,
- C4 is implied from I4 and C2.

We have also the UM-rules C5-C6 which can be deduced as follows:
- C5 is implied from I5 and C1 and from I2 and C2,
- C6 is implied from I5 and C3, I1 and C4, also from I3 and C5.

C1 : R[m? ≠ m%]: “L modify? R” and “L modify% R” => Incompatibility
C1 : R[m? ≠ m%]: “L modify? R” and “L modify% R” => Incompatibility
C2 : R[u? ≠ u%]: “L use? R” and “L use% R” => Incompatibility
C3 : R[m! ≠ m%]: “L modify! R” and “L modify% R” => Incompatibility
C4 : R[u! ≠ u%]: “L use! R” and “L use% R” => Incompatibility
C5 : R[m? ≠ u%]: “L modify? R” and “L use% R” => Incompatibility
C6 : R[m! ≠ u%]: “L modify! R” and “L use% R” => Incompatibility

5.5. Soundness and Completeness Results

Note that R1-R3, R6-R9, I1-I5 and F1-F16 derive new UM-relations from existing UM-relations, while R4-R5 and C1-C6 detect incompatibilities between UM-relations. We will use the symbol wrt for “with regard to”. We will also use “logically” to mean “by using reasoning based on 1st-order logic.

Proposition 5.1 (Preservation of “well-formed”) Each of the UM-rules I1-I5 and F1-F16 derives a well-formed UM-relation when its left hand side member (one or two UM-relations) is well-formed. (Well-formed is defined in Section 4.3.).
Definition 5.1 (Soundness): Consider a set $R$ of rules applicable to UM-relations. $R$ is said sound (implicitly wrt 1st-order logic), if for every set $K$ of UM-relations, all UM-relations and incompatibilities between UM-relations that can be deduced by $R$ from $K$ can also be deduced logically. Intuitively, soundness of $R$ is that $R$ is a subset of the 1st-order logic.

Definition 5.2 (Completeness wrt rules): Consider two sets $F$ and $R$ of rules applicable to UM-relations. $R$ is said complete wrt $F$, if for every set $K$ of UM-relations, all UM-relations and incompatibilities between UM-relations that can be deduced by $F$ from $K$ can also be deduced by $R$. Intuitively, completeness of $R$ wrt $F$ is that $F$ is a subset of $R$.

Definition 5.3 (Completeness): Consider a set $R$ of rules applicable to UM-relations. $R$ is said complete if it is complete wrt 1st-order logic. Intuitively, $R$ is complete if it implies all the UM-relations and incompatibilities between UM-relations that can be implied logically (i.e. by 1st-order logic).

Proposition 5.2 (Soundness): The set of UM-rules $\{I_1-I_5, F_1-F_{16}, C_1-C_6\}$ is sound.

Proposition 5.3 (Completeness wrt $R_1-R_9$): The set of UM-rules $\{I_1-I_5, F_1-F_4, F_9-F_{12}, C_1-C_2\}$ is complete wrt $R_1-R_9$.

5.6. Discussion

5.6.1. Relation of soundness and completeness with FI detection

Soundness is stated in Proposition 5.2 for $\{I_1-I_5, F_1-F_{16}, C_1-C_6\}$, while Proposition 5.3 states completeness of only a subset of $\{I_1-I_5, F_1-F_{16}, C_1-C_6\}$. The question is: Why soundness and completeness are not stated for the same set of UM-rules?

Or more precisely: Why soundness is stated for $\{I_1-I_5, F_1-F_{16}, C_1-C_6\}$ while it can be stated for the subset $\{I_1-I_5, F_1-F_4, F_9-F_{12}, C_1-C_2\}$ which is proved to be sound and complete?

Our answer is developed in the following paragraph.

In fact, we can use uniquely the set of UM-rules $\{I_1-I_5, F_1-F_4, F_9-F_{12}, C_1-C_2\}$ and base our FI detection on this set. The problem is that we have
realized by experience that the UM-rules I₁-I₅ may imply much more UM-relations than what is necessary for our FI detection. Hence, there is the risk to undermine significantly the efficiency of our FI detection procedure. By combining the results of Sections 5.3-5.4, it is easy to see that F₅-F₈, F₁₃-F₁₆ and C₃-C₆ are implied by combining I₁-I₅ with \{F₁-F₄, F₉-F₁₂, C₁-C₂\}. Interestingly, we have realized by experience that I₁-I₅ are indeed relevant for our FI detection only to be combined with \{F₁-F₄, F₉-F₁₂, C₁-C₂\} to derive what can be derived by the missing UM-rules F₅-F₈, F₁₃-F₁₆ and C₃-C₆. Our strategy is therefore to adapt the complete set \{I₁-I₅, F₁-F₄, F₉-F₁₂, C₁-C₂\} by removing I₁-I₅ and adding F₅-F₈, F₁₃-F₁₆ and C₃-C₆. We obtain \{F₁-F₁₆, C₁-C₆\} which is the set of UM-rules which are used for our FI detection. Intuitively, this is equivalent to using the complete set \{I₁-I₅, F₁-F₁₆, C₁-C₆\}, but by applying I₁-I₅ only to derive UM-relations which may be relevant for our FI detection.

5.6.2. About soundness of R₁-R₉

Completeness stated by Proposition 5.3 is wrt R₁-R₉. Intuitively, every UM-relation and incompatibility between UM-relations that is implied from R₁-R₉ can also be implied from \{I₁-I₅, F₁-F₄, F₉-F₁₂, C₁-C₂\}. A question that arises is: Is \{I₁-I₅, F₁-F₈, C₁-C₄\} complete (Def. 5.3)? The answer to this question is Yes if R₁-R₉ is complete. Hence, another question that arises is: Is R₁-R₉ complete? At the present time, we have not a formal answer to this question, but it is worth noting that our development of R₁-R₉ has been dictated by the desire to obtain a sound set of rules which is as much complete as possible. Let us give some explanations to clarify this aspect. Recall that UM-relations are based on:

1) actions “use” and “modify”, and
2) characterizing each action by “always”, “sometimes”, “never” or “maybe”.

The development of R₁-R₉ has been dictated as follows:

- R₁-R₂ are related to point 1: they targets to specify as much as possible the distinction between actions “use” and “modify”.
- R₃-R₅ are related to point 2: they target to specify as much as possible the distinction between “always”, “sometimes” and “never”. “maybe” is not considered because it is a too coarse information which does not permit any deduction.
- R₆-R₉ target to derive logically new UM-relations by combining existing UM-relations.
R6-R7 consider the cases where an action is followed by “use!” or “modify!”, while R8-R9 consider the cases where an action is followed by “use?” or “modify?”.

The two cases are distinguished because R8-R9 are too coarse since they imply a UM-relation with an action “use#” or “modify#” (see Section 5.1.3).

5.6.3. Utility of use# and modify#

One may wonder why “maybe” characterization (use#, modify#) has been used although it represents a too coarse information. In fact, a UM-relation “L x# R” (where x is “use” or “modify”) is clearly irrelevant if there exists a UM-relation with the same L, R and x, but where x is characterized by !, ? or % instead of #. For example, “A use# B” is irrelevant if we have “A use? B”, “A use! B” or “A use% B”. Otherwise, we will see in Sect. 6 that “L x# R” may be relevant in FI detection to model a suspected FI.

5.7. Example of Using UM-rules to Derive new UM-Relations

Example 6: Consider Example 4 of Sect. 4.4 and apply some UM-rules to the UM-relations M1-M9. We obtain the following UM-relations that enrich the UM-model of Lender.

Applying F3 to M6 and M7: Lender modify! Lender.amount
Applying F4 to M8 and M1: Lender.quote() modify? Approver.amount to M9 and M3: Lender.quote() modify? Assessor.amount
Applying F9 to M6 and M8: Lender use# Approver.approve() to M6 and M9: Lender use# Assessor.assess()
Applying F4 to M9 and M4: Lender.quote() modify? Assessor.risk
Applying F12 to M8 and M2: Lender.quote() modify# Approver.rate to M9 and M5: Lender.quote() modify# Assessor.rate

In this example, the suspected accesses (use#, modify#) deduced from F9 and F12 are effective, hence we have a more accurate model if we replace “use#” by “use?” and “modify#” by “modify?”. We have not shown the influence of conditions in the application of rules; we will illustrate their influence in FI detection in Sects. 7.1 and 7.5.
6. FI Detection Method Based on UM-Relations

As already mentioned, there exist many FI detection methods with a high power of detection, but which are prone to state space explosion. In this section, we propose an FI detection method that reduces this problem while keeping an acceptable power of FI detection. The approach is off-line and consists in detecting FIs in a WS during its design (from scratch or by composing existing WS). More precisely, the approach consists in constructing a UM-model of the WS under design, and then in analyzing such a UM-model to detect FI patterns which correspond to symptoms of FI. The designer is informed about each detected symptom and should check if it corresponds to an effective FI. This necessity of the intervention of the designer implies that the FI detection procedure is not completely automatic. This is the price to pay to reduce the state space complexity.

The proposed FI detection method consists of three steps. The first step is to construct a UM-model of the WS under design. The second step is to check if the UM-model is well-formed \((i.e.\) all its UM-relations are well-formed) and to enrich it. The third step is to analyze the UM-model to detect symptoms of FIs. The three steps are presented in Sections 6.1-6.3 respectively.

**Definition 6.1** \((F\text{-relevance})\) A pair of UM-relations is said \(F_{1-8}\)-relevant if it can be a left hand side member of a fusion UM-rule of \(F_{1-8}\). A pair of UM-relations is said \(F_{9-16}\)-relevant if it can be a left hand side member of a fusion UM-rule of \(F_{9-16}\).

6.1. Step 1: UM-Model Construction

Let \(S\) be the WS under design. The first step can be skipped if the UM-model of \(S\) already exists and is given as input to the second step (Section 6.2). Otherwise, we have the following two different cases, which are presented in Sections 6.1.1 and 6.1.2 respectively:

- \(S\) is designed from scratch, .i.e. \(S\) is a feature (see Sect. 4.1) ;
- \(S\) is designed by composing several given WSs \(S_1, S_2, \ldots, S_n\).
6.1.1. Step 1 when $S$ is designed from scratch

We consider that the designer has defined on paper the UM-model of $S$. The first level of the model is an interface model (Section 4.1) which consists of a class with empty methods. Since $S$ is a feature (basic WS), the attributes of the class are basic. The second level is a UM-model consisting of UM-relations “$L \times R$”, where $L$ and $R$ are object(s) of the class defined in the first level, or attributes or methods of that object(s), as shown in Sections 4.2-4.5. The designer edits the UM-model, for example with any text editor or some UM-editor which is specifically designed to edit interactively UM-models.

6.1.2. Step 1 when $S$ is designed by composing WSs $S_1$, $S_2$, ..., $S_n$

We consider that UM-models $S_1$, $S_2$, ..., $S_n$ are given as inputs of Step 1, for example in text files. The designer has access to these UM-models, for example with any text editor or some specific UM-editor. With the available editor, the designer has to construct a UM-model which merges the UM-models of $S_1$, $S_2$, ..., $S_n$. Some treatments may have to be done in the obtained UM-model. Typically, a treatment consists in removing, adding and/or replacing a UM-relation. The treatment is for example used to model the coordination of the composed WSs. The result of merging and treatment is the UM-model of $S$. To understand the necessity of treatment, consider for example the composition of two WSs $S_1$ and $S_2$ by choreography. Each of $S_1$ and $S_2$ may have to call methods of the other one. Hence, the composition of $S_1$ and $S_2$ may require that the designer applies some modifications to $S_1$ and/or $S_2$ by removing, adding and/or replacing some UM-relation(s). The modified UM-relations should be indicated (e.g. by a flag), because the modifications may be a cause of FI and thus should be considered in the phase of FI detection (in Section 6.3, Step 3, Pattern 3).

6.2. Step 2: Verifying that the UM-Model of $S$ is well-formed and Enriching it

This step is divided in the following three substeps which will be explained and justified in their corresponding sections:

- **Substep 2a**: checking if each UM-relation of the UM-model is well-formed, as specified in Section 4.3;
- **Substep 2b**: enriching the UM-model by applying the UM-rules F₁-F₈ of Sections 5.3;
- **Substep 2c**: enriching the UM-model by applying the UM-rules F₉-F₁₆ of Sections 5.3.

Substeps 2b and 2c are used separately, because F₉-F₁₆ derive UM-relations with actions use# and modify# and require a specific treatment as explained in Section 5.6.3.

Substeps 2a-2c will be illustrated by the following example of UM-model, where the UM-relations are identified by rᵢ, m() is a method, and v is a basic attribute:

<table>
<thead>
<tr>
<th>r₁</th>
<th>U   use!</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>r₂</td>
<td>U   use%</td>
<td>V</td>
</tr>
<tr>
<td>r₃</td>
<td>V   modify!</td>
<td>W</td>
</tr>
<tr>
<td>r₄</td>
<td>U   modify%</td>
<td>W</td>
</tr>
<tr>
<td>r₅</td>
<td>X   use?</td>
<td>U</td>
</tr>
<tr>
<td>r₆</td>
<td>W   use?</td>
<td>Z</td>
</tr>
<tr>
<td>r₇</td>
<td>X   use!</td>
<td>Z</td>
</tr>
<tr>
<td>r₈</td>
<td>L   modify?</td>
<td>m()</td>
</tr>
<tr>
<td>r₉</td>
<td>v   use!</td>
<td>R</td>
</tr>
</tbody>
</table>

### 6.2.1. Substep 2a: verifying if all UM-relations are well-formed

The constraints specified in Section 4.3.1 are checked for each UM-relation “L use R”, and the constraints specified in Section 4.3.2 are checked for each UM-relation “L modify R”. At the end of the procedure (outlined below), the returned set X contains all UM-relations detected as non-well-formed. We consider that the subsequent steps cannot be executed while the returned X is not empty. Hence, when X is not empty, the designer must correct the non-well-formed UM-relations and re-execute Substep 2a until the returned X is empty. As in Step 1 (Section 6.1.2), the correction should be indicated (*e.g.* by a flag), because it may be a cause of FI and thus should be considered in the phase of FI detection (in Section 6.3, Step 3, Pattern 3).

**Procedure to find the non-well-formed UM-relations**

**Input:** R = set of UM-relations obtained after Step 1

**Result:** X = set of non-well-formed UM-relations of R

**BEGIN**
X := empty set  
for each UM-relation A of R :  
|  if (A is non-well-formed)  
|   | insert A in X  I have not used “move A to X” because I do not want to remove A from R  
|  end-if  
end-for  
return X  
END

For our example, the UM-rule $r_8$ is non-well-formed because in “L modify R”, R can be a basic or complex attribute, it cannot be a method (see Section 4.3.2). A method is used (by calling it), it cannot be modified. Another non-well-formed UM-rule is $r_9$ because in “L x R”, L cannot be a basic attribute; the latter can be used or modified, it cannot use or modify. We consider here that the adopted solution is to remove the non-well-formed $r_8$ and $r_9$. This may require adapting the WS $S$ under design.

6.2.2. Substep 2b: enriching the UM-model by applying $F_1$-$F_8$

In Section 5.6.1, we have explained why we will use only the set of UM-rules $\{F_1$-$F_{16}, C_1$-$C_6\}$. In fact, the present substep 2b uses $F_1$-$F_8$, while substep 2c uses $F_9$-$F_{16}$. The UM-rules $C_1$-$C_6$ will be used in Step 3, more precisely in Pattern 5 of Section 6.3.

The UM-model $R$ of $S$ is enriched “maximally” by synthesizing all the new UM-relations that are implied by the UM-rules $F_1$-$F_8$. By “maximally”, we mean “iteratively until no new UM-relation is derived”. This can be realized by a fix-point method which iterates the UM-rules $F_1$-$F_8$ until no new UM-relation is generated. The method converges because of the finite numbers of rules ($F_1$-$F_8$) and actions (use!, use?, use%, modify!, modify?, modify%). The structure of the iterative method is shown below.

Explanations of the procedure below: Its input $R$ is the current set of UM-relations. Its result is an enriched $R$, i.e. $R$ with additional UM-relations derived by applying fusion UM-rules $F_1$-$F_8$. $F$ contains the set of $F_{1,8}$-relevant pairs of UM-relations of $R$ which have not yet been treated as a left hand side member of a fusion UM-rule to derive a new UM-relation. Hence, $F$ is initialized as the set of all $F_{1,8}$-relevant pairs of UM-
relations of R. The simplest approach to construct F consists in considering every UM-relation A of R and comparing it with every other UM-relations B of R to determine if the pair (A, B) is the left hand side member of a fusion UM-rule in F₁-F₈. If yes, the pair is inserted in F. The while-loop generates all the new UM-relations that can be derived by applying the UM-rules F₁-F₈ to the pairs of UM-relations of F. At each while-iteration, we select some pair of F, and the objective is to apply the fusion UM-rule F that has as left hand side member. Let B be the UM-relation derived by F. If B is not already in R, it is inserted in R (because R must contain all derived UM-relations). The for-loop consists in updating F by comparing B with every other UM-relation U of R and to insert the pair (B,U) in F if it is F₁₈-relevant (to treat (B,U) in a subsequent while-iteration, as a left hand side member of a UM-rule to try to derive a new UM-relation). Then, the pair is removed from F when it has been treated.

Procedure to enrich the UM-model R by using F₁-F₈:
Input: R = set of UM-relations obtained after Substep 2a
Result: Enriched R
BEGIN
F := set of all F₁₈-relevant pairs of UM-relations of R
while (F is not empty):
| select some pair of F
| let F be the fusion UM-rule having as left hand side member
| let B be the UM-relation which is the right hand side member of F
| if (B is not in R):
| | insert B in R
| | for every UM-relation U in R
| | | if (B,U) is F₁₈-relevant: insert (B,U) in F
| | end-for
| end-if
| remove the pair from F
end-while
END

For our example, the UM-relations r₈ and r₉ were removed in Step 2a and the UM-model R after Substep 2a is \{ r₁, ..., r₇ \}. The set of F₁₈-relevant pairs is F = \{(r₁, r₃), (r₅, r₁)\}. The two pairs of F are left hand
side members of UM-rules $F_3$ and $F_2$ respectively. Let us execute the procedure to this example.

1st iteration: by applying $F_3$ to $(r_1, r_3)$, the following new UM-relation $r_{10}$ is derived:

$$r_{10}: "U \ modify! \ W"$$

$r_{10}$ is inserted in $R$; its addition implies the new $F_{1-8}$-relevant pair $(r_5, r_{10})$ which is inserted to $F$. The treated pair $(r_1, r_3)$ is removed from $F$. Hence, we obtain $R = \{r_1, \ldots, r_7, r_{10}\}$ and $F = \{(r_5, r_1), (r_5, r_{10})\}$.

2nd iteration: by applying $F_2$ to $(r_5, r_1)$, the following new UM-relation $r_{11}$ is derived:

$$r_{11}: "X \ use? \ V"$$

$r_{11}$ is inserted in $R$; its addition implies the new $F_{1-8}$-relevant pair $(r_{11}, r_3)$ which is inserted in $F$. The treated pair $(r_5, r_1)$ is removed from $F$. Hence, we obtain $R = \{r_1, \ldots, r_7, r_{10}, r_{11}\}$ and $F = \{(r_5, r_{10}), (r_{11}, r_3)\}$.

3rd iteration: by applying $F_4$ to $(r_5, r_{10})$, the following new UM-relation $r_{12}$ is derived:

$$r_{12}: "X \ mod? \ W"$$

$r_{12}$ is inserted in $R$; its addition implies no new $F_{1-8}$-relevant pair. The treated pair $(r_5, r_{10})$ is removed from $F$. Hence, we obtain $R = \{r_1, \ldots, r_7, r_{10}, r_{11}, r_{12}\}$ and $F = \{(r_{11}, r_3)\}$.

4th iteration: by applying $F_4$ to from $(r_{11}, r_3)$, the existing UM-relation $r_{12}$ is derived. The treated pair $(r_{11}, r_3)$ is removed from $F$ which becomes empty, and hence the while-loop terminates. We obtain $R = \{r_1, \ldots, r_7, r_{10}, r_{11}, r_{12}\}$.

6.2.3. Substep 2c: enriching the UM-model by applying $F_9-F_{16}$

We proceed with a similar procedure as in Substep 2b, except that:

- we consider UM-rules $F_9-F_{16}$ instead of $F_1-F_8$;
- every new derived UM-relation “$L \ use# R$” or “$L \ modify# R$” is removed if the UM-model of $S$ contains a UM-relation with the same $L$, $R$ and $x$, but where $x$ is characterized by !, ? or % instead of # (see Section 5.6.3).

For our example, the set of UM-relations after Substep 2b is $R = \{r_1, \ldots, r_7, r_{10}, r_{11}, r_{12}\}$. The set of $F_{9-16}$-relevant pairs is $F = \{(r_3, r_6)\}$, where $(r_3, r_6)$ is a left hand side member of the UM-rule $F_{13}$. Let us execute the procedure to this example.
1st iteration: by applying F_{13} to (r_3, r_6), the following new UM-relation r_{13} is derived:

r_{13}: “V use# Z”

r_{13} is inserted in R, its addition implies no new F_{9-16}-relevant pair. The treated pair (r_3, r_6) is removed from F which becomes empty, and hence the while-loop terminates. R_{13} is not removed from R because R contains none of “V use! Z”, “V use? Z” and “V use% Z”. Hence, after Step 2c we obtain R = \{ r_1, \ldots, r_7, r_{10}, \ldots, r_{13}\}.

6.3. Step 3: FI Detection

Step 3 is the proper FI detection procedure. We have identified six FI patterns that represent symptoms (hence potentiality) of FIs. The procedure of Step 3 searches FI patterns in the UM-model R obtained in Step 2, and informs the designer about every detected FI pattern to draw his attention on the corresponding suspected FI. The designer should then react by making adequate verifications. The six identified FI patterns are presented below. For each FI pattern, we indicate a typical reaction of the designer to determine whether the FI is effective or not.

Pattern 1. There exists a “reflexive” UM-relation “a() use! a()” or “a() use? a()” or “a() use# a()”, where a() is a method. This is a symptom of looping behavior which is illustrated by the example of Section 7.1.

Reaction of the designer: the designer should check whether there is an effective looping behavior with action a():

Pattern 2. There exist UM-relation(s) that “modify” and possibly “use” the same entity. That is, two or more UM-relations “K m R” and “L n R” are detected, where m is any “modify*” other than “modify%”, and n is any “use*” or “modify*” other than “use%” and “modify%”. This is a symptom of resource conflict or race condition which is illustrated by the examples of Sections 7.4, 7.7, 8.1, 8.2, 8.3.

Reaction of the designer: the designer should check whether there exists an effective conflicting access to R.

Pattern 3. There exist UM-relation(s) obtained (in Step 1 and/or Step 2a) by correcting (removing, adding and/or replacing) UM-relation(s) of S_1, \ldots, S_n. There is hence the possibility that an identified correction may violate requirements of S_1, \ldots, S_n the designer has in mind,
hence the potentiality of FI. This case is illustrated by the example of Sect. 7.2.

Reaction of the designer: the designer should check whether the identified corrections violate requirements.

Pattern 4. There exist UM-relation(s) with restrictions. By the generic term “restriction”, we mean any of the following two situations:
- There exist UM-relations “L use? R” or “L modify? R” which are associated to specified conditions (Section 4.5).
  Reaction of the designer: the designer should check that the specified conditions are respected.
- There exist UM-relation(s) “L use% R” or “L modify% R”.
  Reaction of the designer: the designer should check that for every “L use% R”, R is effectively never used by L; and for every “L modify% R”, R is effectively never modified by L.

The two sub-cases of Pattern 4 are illustrated in Section 7.5 with use? and modify%.

Pattern 5. There exist incompatible UM-relations. We have actually two types of incompatibilities:
- Two UM-relations “A use* p()” and “A use* q()”, where * may be “!” or “?”, and p() and q() are methods which are incompatible with each other. Here, we assume that in the UM-model R, the designer has specified pairs of incompatible methods.
  For example, this can be formally expressed as follows: for each method p() having incompatible methods, we specify the set \{q_1(), q_2(), …\} of methods which are incompatible with p() by:
  \[\text{Incompatible}[p()] = q_1(), q_2(), \ldots\]
  This case is illustrated by the example of Section 7.3.
- Two UM-relations which are incompatible by the contradiction UM-rules C1-C6 (Section 5.4).
  Incompatibilities are symptoms of inconsistent behavior.
  Reaction of the designer: the designer should check that any detected incompatibility really exists.

Pattern 6. Forbidden UM-relation(s) are present or mandatory UM-relation(s) are missing. Here, we assume that in the UM-model R, the designer has specified forbidden UM-relation and mandatory UM-relations. For example, this can be formally expressed as follows:
Each mandatory (resp. forbidden) UM-relation is followed at its right by the keyword Mandatory (resp. Forbidden). This case is illustrated by the examples of Sections 7.2 and 7.6.

**Reaction of the designer:** the designer should check whether the detected forbidden UM-relations really occur, and whether the missing mandatory UM-relations really do not occur.

Note that we consider only FI detection and not FI resolution. As we have shown, when an FI is detected and reported to the designer, his reaction is to determine if the FI is effective. A further step (left for future work) is to determine how to correct the UM-model to eliminate the detected FIs.

### 6.4. Results and Discussion on Computational Complexity

The development of the UM-based FI detection method has been motivated by the desire to reduce state space explosion. The approach has been that instead of modeling a feature or WS exhaustively by representing many of its states and transitions, we model only certain of its behaviors and properties that are judged relevant. Those relevant behaviors and properties are in the form of UM-relations which themselves are based on objects and their attributes and methods. Two questions arise:

a) How to identify relevant behaviors and properties?

b) How to quantify the reduction of complexity by this approach?

Point a) requires designers who have much experience in designing web services or more generally software services. The designers must also have a good knowledge of the specifications of the WS under design. We have used “designers” in the plural because we think that a good approach to guarantee a good estimation of relevant behaviors and properties is the well-known principle of *diverse design*. The principle is that the same specification of the WS under design is given to several teams who proceed independently to design different versions of UM-models of the WS. Then, the resulting multiple versions are compared with each other to detect their differences. Finally, the teams discuss with each other to agree on a common UM-model. A good example of successful application of diverse design can be found in [36] for firewall design.
About Point b), we have studied the computational complexity of the three steps of FI detection (of Sections 6.1-6.3). The obtained results are given by the following proposition (its proof is in Section 10.4).

**Proposition 6.1 (Complexity of the three steps of FI detection):**
Let \( S_1, \ldots, S_n \) be the WSs to be composed and \( nbR_1, \ldots, nbR_n \) be the sizes (i.e. numbers of UM-relations) of their respective UM-models.

- The computational complexity of Step 1 is in \( O(nbR_1 + \ldots + nbR_n) \).
- The computational complexity of Step 2 is in \( O((nbR_1 + \ldots + nbR_n)^6) \).
- The computational complexity of Step 3 is in \( O((nbR_1 + \ldots + nbR_n)^4) \).

In the case of a single WS (i.e., WS designed from scratch), the above results hold by taking a single \( nbR \) instead of a sum \( nbR_1 + \ldots + nbR_n \).

Let us discuss the results of Proposition 6.1 in comparison to the complexities obtained with more exhaustive models such as those based on automata.

- The exponents 4 and 6 in some results of Prop. 6.1 may seem excessive, but it is worth noting that these are theoretical upper bounds which are very far from the concrete results we have obtained in real examples. The latter are not higher than \( O((nbR_1 + \ldots + nbR_n)^2) \). Even in the theory, it may be impossible to reach complexity with exponents 4 and 6, because our complexity study has been quite permissive as it can be seen in the proof of Prop. 6.1.

- About a basic WS, i.e. not composed of other WSs: with our experience, we expect that the size of an automaton modeling a basic WS should be at least 10 times higher than the size of a UM-model of such a basic WS.

- About a complex WS, i.e. composed of several WSs: the sizes of the composed UM-models are *summed*, instead of being *multiplied* as it is the case with automata-based models. Such a multiplication is the main cause of the well-known state space explosion problem.

**7. Demonstration of FI Detection in the Benchmark of [13] and in an Example of [14]**

Let us demonstrate our FI detection method in the examples of the benchmark of [13]. The latter contains the case study of a fictitious virtual
Ahmed Khoumsi and Zohair Chentouf

A bookstore on which is constructed a benchmark of eight FIs. The following individual WSs are defined:

- **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 WSs *Amazin*, *Supplier* and *Customer* are constructed from the above individual WSs. *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.

### 7.1. Example 1 of [13]: Called “OrderProcessing – OrderProcessing”

The FI manifests itself by a blocking situation in the following way. An order is sent to *Supplier*₁ (by calling a method *order()* of *Supplier*₁) who forwards the order to *Supplier*₂ (by calling a method *order()* of *Supplier*₂) because his stock is empty. Then, *Supplier*₂ in turn decides to forward the order to *Supplier*₁ (by calling a method *order()* of *Supplier*₁) because his stock too is empty too. Hence, we reach the blocking situation where each supplier is waiting the reception of the ordered book from the other supplier. Let us see how our FI detection method detects such FI. The UM-models of *Supplier*₁ and *Supplier*₂ contain respectively the following UM-relations with conditions, as seen in Section 4.5:

- **UM1**: “*Supplier*₁.*order()* use? *SUPPLIER.*order()” : [SUPPLIER not comprising *Supplier*₁],
  
- **UM2**: “*Supplier*₂.*order()* use? *SUPPLIER.*order()” : [SUPPLIER not comprising *Supplier*₂].

The UM-models models of *Supplier*₁ and *Supplier*₂ are composed (Step 1) and the resulting UM-model is enriched (Step 2). In Step 2c, the UM-rule F₁₀ is applied to UM1 and UM2, but after setting *SUPPLIER* of UM1 and UM2 to *Supplier*₂ and *Supplier*₁, respectively; we obtain:

- **UM1-UM2**: “*Supplier*₁.*order()* use# *Supplier*₁.*order()”.
Hence, FI pattern 1 is detected in Step 3. Note that this scenario can be
generalized to a loop involving more than two suppliers: Supplier\(_1\) is
waiting Supplier\(_2\) who is waiting Supplier\(_3\) … Supplier\(_k\) who is waiting 
Supplier\(_1\).

7.2. Example 2 of\cite{ref13}: 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), then the process 
payment is shortcut. Hence, the order is completed without payment. Let 
us see how our FI detection method detects such an FI. Supplier and 
Caching WSs are specified by a set of UM-relations. Consider a method 
\texttt{completeOrder()} which is called in Supplier when everything is ready to 
start payment and delivery processes. The payment process starts by 
calling a method \texttt{pay()}. A UM-relation which is particularly relevant in 
this example is: \texttt{completeOrder()} use! \texttt{pay()}

The UM-models of Supplier and Caching are composed (Step 1) and 
the resulting UM-model is enriched (Step 2). This example illustrates the 
situation where composing two WSs requires that the designer modifies 
the process payment of Supplier as explained above. The present 
composition has the effect to replace the call of a method \texttt{pay()} by a 
conditional call. Hence the above UM-relation is replaced by the UM-
relation \texttt{completeOrder()} use? \texttt{pay()}” (i.e., “use!” replaced by “use?”). 
Hence, FI pattern 3 is detected in Step 3.

Another way to detect the FI is that the designer specifies the UM-
relation “\texttt{completeOrder()} use! \texttt{pay()}” as mandatory. The FI is deduced by 
the fact that the composition has modified this mandatory UM-relation. 
Hence, the FI pattern FI pattern 6 is detected in Step 3.

7.3. Example 3 of \cite{ref13}: 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 FIs are referred to as 
(a) and (b) as follows:

(a) FI Called “Order Processing – Delivery” in \cite{ref13}: The FI manifests 
itself when, due to timing errors, a process payment is aborted while 
the delivery is completed (instead of being aborted). Hence, the
possibility to receive a book which has not been paid (as in Example 2, but for a different reason).

(b) FI Called “Order Processing - Process Payment” in [13]: The FI manifests itself when, due to timing errors, a delivery is aborted while the process payment is completed (instead of being aborted). Hence, the possibility to pay for a book which is not received.

Let us see how our FI detection method detects such FIs. A supplier WS is composed of several features such as: ProcessPayment, Delivery, and OrderProcessing, each one being described by UM-relations. The different UM-models are composed (Step 1) to obtain a UM-model of Supplier which is enriched (Step 2).

The UM-model of Supplier uses the following methods: abortOrder() is called to abort the current order, pay() is called to start payment for the ordered product, and deliver() is called to start delivery of the ordered product. abortOrder() is incompatible with deliver() and pay(), because payment and delivery must not be done when an order is aborted. We assume that the designer has specified these incompatibilities.

The UM-model contains the following three UM-relations:

Hence, the FI pattern 5 is detected in Step 3 for the pairs (R1,R2) and (R1,R3). The incompatible pair (R1, R2) corresponds to FI (a), and the incompatible pair (R1, R3 corresponds to FI (b).

7.4. Example 4 of [13]: Called “Order Processing - Fulfill Order”

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. Let us see how our FI detection method detects such FI. The UM-model and Steps 1 and 2 are as in Example 3 (Section 7.3). After steps 1 and 2 The UM-model of Supplier uses two methods orderProcessing() and fulfillOrder() that modify an attribute price, i.e. we have the following UM-relations:

“orderProcessing() modify? price” “fulfillOrder() modify? price”

Hence, FI pattern 2 is detected in Step 3.
7.5. **Examples 5, 6, 7 of** [13]**: All Associated to Access Profile**

We consider Examples 5, 6 and 7 together, 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 [13]): an untrusted supplier accesses some information in the profile of the customer.

- **In example 6** (called “Access Profile - Access profile” in [13]): 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 [13]): a supplier accesses some information in the profile of the customer when the latter is not connected.

After Steps 1 and 2, the resulting UM-model contains UM-relations such as:

- “Supplier use? profile” : \{Supplier is authorized\}
- “Supplier modify% profile”

Hence, FI pattern 4 is detected in Step 3. Note the condition \{Supplier is authorized\} associated to the first UM-relation, which models the fact that only the authorized suppliers can read a user profile. The “modify%” corresponds to the restriction specifying that no supplier is authorized to modify a user profile. Hence, the designer should check if these restrictions are respected. The FIs of Examples 5, 6 and 7 are due to the non-respect of some authorizations.

7.6. **Example 8 of** [13]**: Called “Order Processing - Order Processing”**

The FI manifests itself by a blocking situation where Supplier₁ is waiting Supplier₂, who in turn is waiting Supplier₁, which corresponds exactly to Example 1 (Section 7.1). Hence Examples 1 and 8 are identical, but in Example 8, the FI is presented with a different viewpoint: *None of the suppliers is available to the other one*. A way to detect this FI is given in Section 7.1. Let us present another way to detect this FI.

We assume that the designer has specified the following UM-relation as forbidden: “Supplier modify? available”, where “available” is a boolean that indicates whether Supplier is available or not. Intuitively, Supplier cannot make himself unavailable.
The fact is that after Steps 1 and 2, the resulting UM-model will contain the above forbidden UM-relation.

Hence, FI pattern 6 is detected in Step 3, which is a symptom that availability changes and hence available can be false in some situations.

7.7. Example of\textsuperscript{[14]}: Called “Spell Checking - Formatting”

The FI manifests itself when the Spell Checker and the Formatter use different languages, e.g., US English and UK English. At the formal level, this FI is similar to the FI of Example 4. In the latter, two methods modify an attribute price. In the present example, two features SpellChecker and Formatter modify an attribute lang specifying the used language. After Steps 1 and 2, the resulting UM-model contains the following UM-relations: “SpellChecker modify? lang” “Formatter modify? lang”.

Hence, FI pattern 2 is detected in Step 3.

8. Demonstration in Detection of Several FIs of\textsuperscript{[15]}

\textsuperscript{[15]} presents an interesting comparative study showing that FIs in Telecom-Services are different from FIs in WSs, and hence FI detection methods developed for the former cannot be easily adapted for the latter. We will apply our FI detection to three types of FIs given in \textsuperscript{[15]}:

- FI between two WSs;
- FI between two Telecom-services;
- FI between a WS and a Telecom-service.

As we will see, the three FIs are related to FI pattern 2 of Step 3.

8.1. FI Between Two WSs of\textsuperscript{[15]}: “Encrypt Information – Payment Information”

The FI manifests itself when the Logging WS uses the encrypted information (purchase order or payment information) while Logging needs to use the information before it is encrypted. After Steps 1 and 2, we obtain UM-relations where an attribute paymentInfo is modified by a method encrypt(), while another method logging() reads the attribute paymentInfo. That is, we have the following UM-relations:

“encrypt() modify! PaymentInfo” “logging() use! PaymentInfo”
Hence, FI pattern 2 is detected in Step 3.

8.2. FI Between Two Telecom-Services of [15]: “Voicemail (VM) – Call Blocking (CB)”

Contrary to previous examples, here we consider Telecom-services instead of WSs. The FI manifests itself when a caller rejected by Call-Blocking (CB) of a callee is able to leave a (potentially unwanted) voicemail via Voicemail (VM). After Steps 1 and 2, we obtain UM-relations where an attribute callStatus is modified by CB (to busy status) and read by VM (busy status is the trigger of VM). That is, we have the following UM-relations:

“CB modify! callStatus” “VM use! callStatus”.

Hence, FI pattern 2 is detected in Step 3.

8.3. FI Between a Telecom-Service and a WS of [15]: “Talk-To-Agent (TTA) – Do-Not-Disturb (DND)”

This is a special case, in the sense that we have a mixed composition, i.e., a WS is composed with a Telecom-service. The FI manifests itself when a customer wants to be joined by an agent to talk with him (WS called TTA), while he has configured the Telecom-service Do-Not-Disturb (DND) to reject all calls. After Steps 1 and 2, we obtain UM-relations where the attribute callStatus (already used in the example of Section 8.2) is modified by DND (to the status busy, for example) and read by a method tta(). That is, we have the following UM-relations: “DND modify! callStatus”, “tta() use! callStatus”. Hence, FI pattern 2 is detected in Step 3.

9. Conclusion

We have developed a method to detect FIs in WSs, which makes a trade-off between reducing state space explosion and increasing the power of FI detection. The proposed method is based on the development of a rigorous Use-modify framework. The latter contains a UM-language to describe WSs at a high abstraction level by objects and UM-relations which indicate uniquely information such as who uses what and who modifies what, and characterize each action “use” or “modify” by “always”, “sometimes”, “never” or “maybe”. Conditions and restrictions may also be associated to UM-relations. In addition to the UM-language,
the UM-framework contains also a set of UM-rules (i.e. rules applicable to UM-relations) that are proved to be sound and complete. The UM-rules permit to derive new UM-relations from existing UM-relations and detect incompatibilities between UM-relations. The developed UM-based FI detection method reports FI symptoms to the designer who then has to verify the effectiveness of the suspected FIs.

We have demonstrated the applicability of our FI detection method in several concrete examples. Indeed, we have applied our method to detect all FIs of the benchmark of [13] and an FI in [14]. We have also applied our method to detect several FIs indicated in [15], where the composed services can be WSs and/or telecommunication services. We think that our FI detection approach can be better than [13] because in the latter many modeling formalisms have to be used: Goal-oriented Requirement Language (GRL), Use-Case Maps (UCM), and Finite State Processes (FSP).

In Section 6.4, we have briefly discussed the gain in computational complexity of our UM-based approach. In a near future work, we plan to study more thoroughly that complexity. For that purpose, we plan to develop a prototype of the UM-based FI detection method to evaluate it more accurately. Another planned future work is to study FI resolution phase, which consists in solving the detected FIs.

10. Proofs

10.1. Proof of Proposition 5.1

We have to prove that the UM-rules I₁-I₅ and F₁-F₁₆ preserve the well-formed property specified in Section 4.3 (for I₁-I₅, see also the explanations in Section 5.2).

The well-formed property is preserved by I₁-I₂ because for any I₁ or I₂, the well-formed property requires stronger constraints on the left hand side of the UM-rule than on its right hand side.

The well-formed property is preserved by I₃-I₄ because for any I₃ or I₄, the well-formed property requires the same constraints on the left and right hand sides of the UM-rule.

The well-formed property is preserved by I₅ because of the condition associated with I₅.

The well-formed property is preserved by F₁-F₄ because for any of F₁ to F₄: The “well-formed” constraints on L are the same in the left and
right hand sides of the UM-rule; and the “well-formed” constraints on \( R \) are the same in the left and right hand sides of the UM-rule.

The well-formed property is preserved by \( F_5 \) (resp. \( F_7 \)) because it is obtained by combining \( I_1 \) with \( F_1 \) (resp. \( F_3 \)) which have just been proved to preserve the well-formed property. In the same way, the well-formed property is preserved by \( F_6 \) (resp. \( F_8 \)) because it is obtained by combining \( I_2 \) with \( F_2 \) (resp. \( F_4 \)) which have just been proved to preserve the well-formed property.

The well-formed property is preserved by \( F_9-F_{16} \) because we can make the same reasoning as with \( F_1-F_8 \).

10.2. Proof of Proposition 5.2

We have to prove that the set of UM-rules \( \{I_1-I_5, F_1-F_{16}, C_1-C_6\} \) is sound. We will use the term “logically” to mean “by using reasoning based on \( \text{1}^{\text{st}} \)-order logic”.

The set of rules \( R_1-R_9 \) is sound because every rule \( R_1 \) to \( R_9 \) has been justified logically in Section 5.1.

In Section 5.2, we have shown that the implication UM-rules \( I_1-I_5 \) are direct translations of rules \( R_1-R_3 \).

In Section 5.3, we have shown that the fusion UM-rules \( F_1-F_4 \) are direct translations of rules \( R_6-R_7 \), and \( F_9-F_{12} \) are direct translations of rules \( R_8-R_9 \).

In Section 5.4, we have shown that the contradiction UM-rules \( C_1-C_4 \) are direct translations of rules \( R_4-R_5 \).

Consequently, the set of UM-rules \( \{I_1-I_5, F_1-F_4, F_9-F_{12}, C_1-C_4\} \) is a direct translation of the set of rules \( R_1-R_9 \). Since \( R_1-R_9 \) is sound, we deduce that its translation \( \{I_1-I_5, F_1-F_4, F_9-F_{12}, C_1-C_4\} \) is sound.

In Section 5.3, we have shown that \( F_5-F_8 \) are derived logically from \( I_1-I_2 \) and \( F_1-F_4 \), and that \( F_{13}-F_{16} \) are derived logically from \( I_1-I_2 \) and \( F_9-F_{12} \).

In Section 5.4, we have shown that \( C_5-C_6 \) are derived logically from \( \{I_1-I_5, C_1-C_4\} \). Since \( F_5-F_8, F_{13}-F_{16} \) and \( C_5-C_6 \) are derived logically from UM-rules of \( \{I_1-I_5, F_1-F_4, F_9-F_{12}, C_1-C_4\} \) which has just been proved to be sound, we have that the whole set \( \{I_1-I_5, F_1-F_{16}, C_1-C_6\} \) is sound.
10.3. Proof of Proposition 5.3

We have to prove that the set of UM-rules \{I_1-I_5, F_1-F_4, F_9-F_{12}, C_1-C_2\} is complete wrt R_1-R_9.

We have shown in Sections 5.2-5.4 and in the proof of Proposition 5.2 that the set of UM-rules \{I_1-I_5, F_1-F_4, F_9-F_{12}, C_1-C_4\} is a direct translation of the set of rules R_1-R_9. Moreover, in Sections 5.2-5.4, the UM-rules \{I_1-I_5, F_1-F_4, F_9-F_{12}, C_1-C_4\} have been obtained by considering all possible translations of R_1-R_9 into UM-rules. In other words, \{I_1-I_5, F_1-F_4, F_9-F_{12}, C_1-C_4\} are the unique possible translations of R_1-R_9 into UM-rules. Therefore, R_1-R_9 and \{I_1-I_5, F_1-F_4, F_9-F_{12}, C_1-C_4\} imply the same UM-relations and detect the same pairs of incompatible UM-relations. Besides, we have seen in Section 5.4 that C_3-C_4 can be implied logically from I_3-I_4 and C_1-C_2. Hence, C_3 and C_4 can be omitted in the study of completeness. Consequently, R_1-R_9 and \{I_1-I_5, F_1-F_4, F_9-F_{12}, C_1-C_2\} imply logically the same UM-relations and incompatibilities between UM-relations. In other words, \{I_1-I_5, F_1-F_4, F_9-F_{12}, C_1-C_2\} is complete wrt R_1-R_9.

10.4. Proof of Proposition 6.1

Let S_1, …, S_n be the WSs to be composed and nbR_1, …, nbR_n be the sizes (i.e. numbers of UM-relations) of their respective UM-models. For simplicity of notation, we use nbR to denote nbR_1+…+nbR_n.

10.4.1 Computational complexity of Step 1

- Merging all UM-relations: its complexity is in the order of the total number of all number of UM-relations, i.e. O(nbR).
- Modifying UM-relations: in the worst case, all UM-relations are modified, which is in the same order as merging, i.e. O(nbR).
- Adding some UM-relations: the number of added UM-relations is typically quite less than the total number of UM-relations, i.e. its order is smaller than O(nbR).

Therefore, we obtain that in Step 1 the computational complexity and the number of UM-relations is in O(nbR).
10.4.2 Computational complexity of Step 2

2a: Check if each UM-relation is well-formed

Its complexity is in the order of the number of UM-relations obtained in Step 1, i.e. $O(nbR)$.

2b. Enriching the UM-model by applying F₁-F₈

Let $|X|$ denote the size (or cardinality) of a set $X$.
Let $R_i$ and $R_f$ be the set $R$ of UM-relations before and after Step 2, respectively (indices $i$ and $f$ are for initial and final). Recall that $O(|R_i|) = O(nbR)$ (result of Step 1).

Each UM-relation of $R_f$ has:
- its left hand side member as a left hand side member of some UM-relation of $R_i$;
- its right hand side member as a right hand side member of some UM-relation of $R_i$.

Hence, at the maximum, for each of the left hand side members of UM-relations of $R_i$, we may associate any of the right hand side members of UM-relations of $R_i$. That is, we may have at the maximum $nbR^2$ UM-relations in $R_f$. Consequently, the size of $R$ after Step 2 is upper-bounded by $O(nbR^2)$, i.e. $O(|R_f|) = O(nbR^2)$.

Let us consider the algorithm that constructs $R_f$ from $R_i$.
Let $F_i$ be the initial $F$ constructed just before the while-loop.

During the execution of this algorithm, we define:
- $q$ as the number of times a UM-relation is inserted in $R$;
- $p$ as the number of times a pair of UM-relations is inserted in $F$;
- $k$ as the number of times a pair of UM-relations is removed from $F$.

We have:
- $q$ is in the order of $|R_f|$, and we have seen that $O(|R_f|) = O(nbR^2)$. Hence, $O(q) = O(nbR^2)$. 


- $F_i$ contains pairs of UM-relations of $R_i$, and we have seen that $O(|R_i|) = O(nbR)$. Hence, $O(|F_i|) = O(|R_i|^2) = O(nbR^2)$.

- Each of the $q$ times where a UM-relation is inserted in $R$, we may have pairs of UM-relations inserted in $F$ (in the for-loop). The number of these pairs is at most in the order of the current size of $R$, which is at most $O(|R_f|)$ which was shown to be $O(nbR^2)$. Hence, $O(p) = O(|R_f| \times q) = O(nbR^4)$, because it has been shown that $O(|R_f|) = O(nbR^2)$ and $O(q) = O(nbR^2)$.

- $|F_i| - k + p = 0$ (i.e. $k = |F_i| + p$), because $F$ is empty at the termination of the algorithm. Since it has been shown that $O(|F_i|) = O(nbR^2)$ and $O(p) = O(nbR^4)$, we conclude that $O(k) = O(nbR^4)$.

We have shown that the number $k$ of iterations of the while-loop is upper-bounded by $nbR^3$. At each of the $k$ iterations of the while-loop:

- the complexity for checking the condition of “if” is upper-bounded by $O(|R_f|) = O(nbR^2)$ because at most, $B$ is compared to every UM-relation of the current $R$. The complexity of all other statements is in $O(1)$.

- The number of iterations of the for-loop is in $O(|R_f|) = (nbR^2)$

Hence, the complexity of the algorithm (i.e. Step 2b) is upper-bounded by $O(nbR^4 \times nbR^2) = O(nbR^6)$.

2c: Enriching the UM-model by applying F9-F16

Applying F9-F16 has its complexity in the same order as that of Step 2b, i.e. upper-bounded by $O(nbR^6)$.

Recall that the size of $R$ after Step 2b (and also Step 2c) is in $O(nbR^2)$.

Removing irrelevant UM-relations:

- Searching UM-relations “L use# R” or “L modify# R”: in $O(nbR^2)$.

- For each found UM-relation: searching a more accurate UM-relation: in $O(nbR^2)$.

Hence, removing irrelevant UM-relations is upper-bounded by $O(nbR^4)$.

Therefore, Step 2c is in $O(nbR^6)$. 
Therefore, the total complexity of Step 2 is upper-bounded by \(O(nbR^6)\).

### 10.4.3 Computational complexity of Step 3

Recall that \(O(nbR^2)\) is the order of the size of \(R\) after Step 2.

We compute the complexity for each pattern:

**Pattern 1:** Detecting “reflexive” UM-relations “\(m() \text{ use } m()\)”, where * \(\text{ is ?}, ! \text{ or } #\) (i.e. * \(\text{ is not } \%\)). It is in the size of \(R\) : \(O(nbR^2)\).

**Pattern 2:** Detecting two or more UM-relations “\(K \text{ m } R\)” and “\(L \text{ n } R\)”, where \(m\) is any “modify*” other than “modify%”, and \(n\) is any “use*” or “modify*” other than “use%” and “modify%”. It is in the square of the size of \(R\) : \(O(nbR^4)\).

**Pattern 3:** Detecting UM-relation(s) modified in Step 1. It is in the size of \(R\) : \(O(nbR^2)\).

**Pattern 4:** Detecting UM-relation(s) with restrictions. It is in the size of \(R\) : \(O(nbR^2)\).

**Pattern 5:** Detecting incompatible UM-relations. Since we have to consider pairs of UM-relations, the complexity is in the square of the size of \(R\) : \(O(nbR^4)\).

**Pattern 6:** Let \(nbM\) and \(nbF\) be the numbers of UM-relations specified as mandatory and forbidden, respectively.

Verifying the presence of mandatory UM-relations: for each mandatory UM-relation \(M\), we go through the UM-relations of \(R\) to verify the presence of \(M\). Hence, the complexity is at most in \(nbM\) multiplied by the size of \(R\), i.e. \(O(nbM \times nbR^2)\).

Verifying the absence of the forbidden UM-relations: for each forbidden UM-relation \(F\), we go through the UM-relations of \(R\) to verify the presence of \(F\). Hence, the complexity is at most in \(nbF\) multiplied by the size of \(R\), i.e. \(O(nbF \times nbR^2)\).

Hence, the total complexity of Pattern 6 is in \(O((nbM + nbF) \times nbR^2)\).
Typically, the total number \( (nbM + nbF) \) of mandatory and forbidden UM-relations is smaller than the size of \( R \), i.e. \( O(nbM + nbF) < O(nbR^2) \). Therefore, the complexity of Pattern 6 is upper-bounded by \( O(nbR^4) \). Hence, the total computational complexity of the three steps: \( O(nbR^6) \).

References

A Use-Modify Framework to Detect Feature Interactions in Web Services


استخدام وتعديل إطار عمل للكشف التفاعلات ذات الخصوصية

في خدمات الويب

أحمد خمسي، وزهر شنتوف*

قسم الإلكترونيات وهندسة الحاسبات، جامعة شيربروك، شيربروك، كندا، و* قسم هندسة البرمجيات، كلية المعلومات وعلوم الحاسبات، جامعة الملك سعود، الرياض، المملكة العربية السعودية

zchentouf@ksu.edu.sa

المستخلص. إحدى فوائد ترکيب خدمات الويب هي الحصول على خدمات جديدة من خدمات موجودة مسبقًا. ولكن ترکيب خدمات الويب قد تكون عرضة لبعض التفاعلات الخاصة والتي تؤدي إلى حدوث تصرفات غير مرغوب فيها. عند استخدام أكثر من خدمة ويب مع بعضها البعض، وقد أضحى معروفًا اليوم أن طرائق اكتشاف التفاعلات ذات الخصوصية تقتضي إلى تعقيد يصعب التحكم فيه دون أن يضعف ذلك من قدرتها على اكتشاف الأخطاء. هدف البحث هو تطوير طريقة لاكتشاف التفاعلات الخاصة في خدمات الويب والتي تهدف إلى تقليل التعقيدات التي يصعب التحكم فيها، بينما نحاول الحفاظ على قدر مقبول من اكتشاف التفاعلات ذات الخصوصية.

الجودة المقترحة تعتمد على استخدام لغة جديدة لنمذجة خدمات الويب بمثابة تجربة عالية. يقدم نموذج الاستخدام والتعديل لخدمات الويب معلومات مثل "من يستخدم ماذا" و"من يعدل ماذا" ويتميز كل عملية استخدام وتعديل بـ "دائماً" و"أحياناً" و"بأي حال". الاستخدام والتعديل يشيران أيضًا إلى كل استخدام وتعديل إذا كانت شروطًا محددة أو غير محددة. درسنا التعقيد الحسابي لطريقتنا للكشف التفاعلات ذات الخصوصية ووضحتنا قابليتها للتطبيق في عدة أمثلة.