

Modeling business process of web services with an Extended STRIPS Operations to detection feature interaction problems runtime

Jiuyun Xu, Kun Chen, Youxiang Duan
School of Computer & Communication Engineering
China University of Petroleum
Dongying, China
Email: jiuyun.xu@computer.org ; ck@126.com;
yxduan@upc.edu.cn

Stephan Reiff-Marganiec
Department of Computer Science
University of Leicester
Leicester, UK
Email: srm13@leicester.ac.uk

Abstract — Service-Oriented Computing achieves its full potential when services interoperate. Current service-oriented computing research is concerned with the low level interoperation among services, such as service discovery, service composition etc. However, a high level research issue in form of the feature interaction problem is challenging the interoperation of services. Traditional feature interaction methods are focused on the service design phase using formal methods or pragmatic software engineering analysis. Autonomy and distribution of service development and deployment create needs for runtime detection and resolution of feature interactions in SOC. This paper investigates the detection of feature interactions in web services at runtime and proposes ESTRIPS, an extended STRIPS operation to ensure conflict-free services are identified to populate business processes, using a combination of OWL-S, SWRL and runtime data extracted from SOAP messages. First, we define the feature interaction problem in business process during their execution and then introduce the ESTRIPS method. The implementation of a prototype is illustrated and a real world scenario shows the plausibility of our method for detecting feature interactions in business processes.

Keywords: *Feature Interaction; Web services; Extended STRIPS Operation; OWL-S; SOAP Message*

I. INTRODUCTION

Service-oriented computing is as buzzword implying amongst others flexible interoperation. Much research work has focused on flexible, dynamic web service discovery, invocation and composition at low-level. The importance of reliability and security of business process is critical for service customers from the industrial viewpoint. Feature interaction, which is an obstacle for the deployment of telecom service in the large scale, also affects the deployment of web services in the service-oriented computing domain. In the telecommunications research community, features as well as services, are units of functionality which are correct on their own, but when used in combination they might influence their behavior in undesired ways. This problem has been known as Feature Interaction, a term coined by Bellcore in the late 1980s. The Feature Interaction (FI) problem [1, 2, 3] has become one of the important bottlenecks for the deployment of new services. FI is not a bug in the implementation of individual services, but an emergent behavior if features are used in conjunction.

Service-oriented architecture (SOA) holds the promise for businesses of allowing for quick adaptation of systems. Web services are a way of encapsulating application functionality in a location and implementation transparent manner. However, if services are composed the potential feature interaction arises. Akin to FI, Web services may interact with each other in unexpected and often undesirable ways negatively affecting service quality and user satisfaction. [4] describes this as Web Services Feature Interaction (WSFI) problem. Other research has focused on feature interactions in service-oriented computing, e.g. [6, 8]. In previous work [15], we have categorized two types feature interaction, which are the side-effects of a business process.

Feature interaction in web services means that one of the services is leading to undesirable behavior for the business customer while the business process is executing, although the web service is behaving as expected in isolation. Generally, feature interaction is not a fault of a service but rather a result of emergent interaction between services. As telecoms markets have traditionally been closed and tightly controlled, the FI problem was manageable due to in house design knowledge, small numbers of features and good availability of working details [5]. As the telecoms market became more open, the need for solutions to FI increased and new challenges were posed. The Web services market has always been open, with many people providing services that are supposed to work seamlessly together. Hence lessons learned in telecoms, should be considered in the context of the WSFI problem. Predominately, the detection and resolution of WSFIs will become important to foster rapid deployment of new services and robustness of composite services.

There has been plenty of work on the prevention, detection and resolution of FI in telecommunication systems [3], but the traditional detection methods are not suitable for the problem in Web services as: (1) web services are not centrally controlled and there is no global understanding of side effects and the operations of the services, and (2) FI in Web services is based on undesirable side effects such as an inconsistent states, or data inaccuracies rather than inconsistent events as is often the case in telecoms. However, web services provide a number of opportunities, most notably that if considering Semantic Web Services a range of information is available that provides insight into the desired actions of web services accessible to all and not just to domain experts.

Hence there is a need for methods that operate at runtime to detect interactions which are caused by services encountering each other in their operation and producing data based side effects that can lead to inconsistencies and violation of assumptions while a service composition is being executed – in some way monitoring interactions with the opportunity to interfere before things go wrong if problems are detected.

The rest of this paper is organized as follows: section 2 introduces related work. Then section 3 provides background on OWL-S and introduces the categories of feature interaction that can occur in web service composition. Section 4 describes our online detection method in detail, including the implementation with some case studies. In sections 5 and 6, we discuss our method and conclude the paper.

II. BACKGROUND

Our detection method is based on the use of logical reasoning and OWL-S. We provide a concise introduction to OWL-S, more information can be found in [13].

OWL-S is the major description language for semantic web services. It is based on an ontology of service concepts that supply a Web service designer with a core for describing the properties and capabilities of a Web service in an unambiguous computer-interpretable form.

OWL-S organizes a service description into four parts: the process model, the profile, the grounding, and the service. The OWL-S process model is most useful for the work presented here, as it provides the required metadata about the Web services.

In our method, we firstly transform process descriptions (mainly *Preconditions*, *inConditions* and *Effects*) into sets of rules expressed in an ontology-aware rule language, namely the Semantic Web Rule Language (SWRL) [13]. This is based on the method presented in [15]. Then we define relevant predicates using the rules to express the composite Web services.

In the web service composition environment, two types of feature interactions are considered. We give the description of two types of feature interaction in detail as follows.

Dissatisfaction of Precondition in the workflow. This type of feature interaction occurs during web service composition execution. The precondition of one web service in the composition is very sensitive to the data of the service execution context. Whenever part of the predicates becomes false due to some previous service's execution changing some value relevant to the input parameters. Generally, this category of feature interaction is occurring during the execution of world altering web services. With logical reasoning, this type of feature interaction can be detected by calculating the precondition of services before they execute.

Conflict among InConditions of web services. This type of feature interaction refers to a conflict between two web services in the workflow caused by interaction of some sensitive data, which business process requesters are considering as almost opposite requirements. For instance, in the telecommunication domain, some value-added services

were implemented as with open APIs, such as the Parlay APIs. The feature interaction between Originating Call Screening (OCS) and Call Forwarding (CF) is an example. When user A registers with the OCS service forbidding calling user C, but permitting to call user B. User B registers with the CF service which forwards all calls to user C. In this context, there is an *Incondition* conflict between these two services. This is also occurring in business processes. For instance, the logistical service needs the address of customers, while some customers do not leave their address to service providers as they see this is privacy violation. This case does not involve all customers but just some of them. So, this kind of feature interaction must be detected during the execution of the web service composition.

We will next illustrate our method to handle the two types of feature interaction during execution of composite web services in detail, but would like to reinforce a difference between this work and planning a web service composition. Clearly when deciding on which services to compose the static description and the semantic description of the Effects and Preconditions is studied. A composition is then planned such that it seems that each service satisfies the requirements of the next in the chain. This planning cannot take into account the runtime interactions mentioned above as these depend on instance data (from the service invocations) rather than on the generic behavior of the services.

III. THE METHOD

Our method is used to detect Web service feature interactions during the execution of the service composition. In this section, we present an overview of the architecture and describe the detection process in detail. A real world scenario shows the example for the detection of the conflict type that exist (the lack of resources to complete a latter part of a workflow and the attempt to invoke a service whose preconditions are not met anymore after an earlier service execution) and are detected.

A. Service Interaction Detection Algorithms

At the beginning, the composite service is in the initial state. After each atomic service within the composite service is executed, we get a new service state, and so on. If the former state is inconsistent with the latter one, or some predicate becomes false we have identified a feature interaction. Figure 1 outlines the detection of feature interactions based on logic reasoning.

Figure 1 provides an overview of the detection process, which consists of six steps as follows:

Step 1. In the first step the SOAP request or SOAP response message of the current service in the workflow execution engine is intercepted, processing is put on hold until a reply message is injected in the system.

Step 2. We extract *Preconditions*, *InConditions* and *Effects* from the OWL-S document using Mindswap OWL-S API [16], which can conveniently read or write OWL-S document. This task is performed by the Feature Interaction Rules Manager. The required data to invoke the

functions is available in the SOAP message as described earlier and is transmitted in this step.

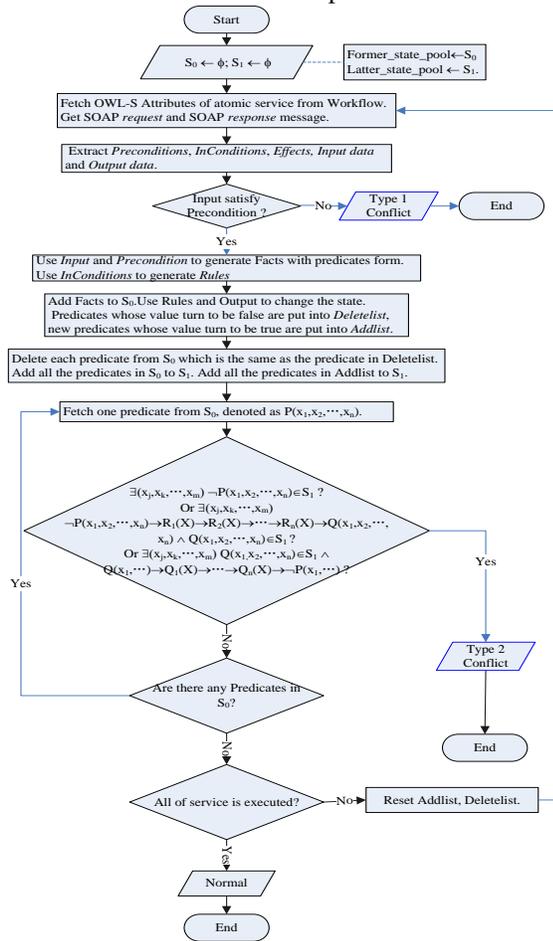


Fig.1. an overview of the detection process

Step 3. This is the first key step in the detection process as it builds the service state information and prepares the extracted data and obtained rules for the detection phase. In more detail, we require two state pools, a *Former_state_pool* and a *Latter_state_pool*. The former denotes the state before the execution of the current service. The latter contains information of the state after executing the service. Before the service is executed, Preconditions and input data from the SOAP request message generate facts (predicates whose values are definitely true); these are put into the *Former_state_pool*. We also maintain two lists, called *Addlist* and *Deletelist* for storing the new predicates during the execution process of the service. The new predicates whose values are definitely true are put into *Addlist*, the predicates whose values change from true to false are put into *Deletelist*. *inConditions* is used to generate FI rules. After the service is executed, FI rules affect the service state and the state is changed according to Effects and the output data from the SOAP response message. In particular, we delete each predicate that occurs in both the *Deletelist* and the *Former_state_pool* from the latter. Then all remaining predicates from the *Former_state_pool* and all the predicates from the *Addlist*

are added to the *Latter_state_pool*. The two state pools now represent two states during the execution process of the service composition which will be evaluated in step 4.

Step 4. In this step we determine whether a Web service feature interaction occurs. There are two cases that can lead to feature interaction (the two types of interaction mentioned earlier). One is that the *Former_state_pool* doesn't satisfy the *Preconditions* of the current service. The other is that there is a conflict between two atomic predicates in the *Former_state_pool* and *Latter_state_pool*. Using the Knowledge Base and the Inference Engine, we identify whether either of the two cases will occur.

Step 5. In this step information on newly detected interactions is recorded in the Interaction Information Base, and the events are recorded to the log. This data helps with future detection.

Step 6. If an interaction is detected, the conflict resolver will be queried to provide a solution. This step will lead to transmitting progress information to the workflow execution engine and allow for the processing of the workflow to continue.

B. Critical Processes

In our method, some information will be extracted from web services during their execution. Combining every OWL-S file of every service with parameters coming from SOAP messages, which are including SOAP request and SOAP response messages, we can obtain the predicates that make up the Prolog file. The detail of process is presented as follows:

During the execution of a composite web service, the system extracts the information from the execution context to detect the two types of feature interaction at runtime.

Firstly, we add predicates in the Prolog file from precondition with the instantiation of parameters from the SOAP request message; If the current web service to be executed is not the first one, we should query the predicates as goal to find whether there is an invalidation of preconditions before the current service execution; this invalidation is the first type of feature interaction.

Secondly, we also add predicates in the query Prolog file from the effect with the instantiation of the parameters from the SOAP response message after one of the web services finishes its execution. In this step, we will add the predicates of the head part of effect rules but delete the predicates of the body part of the effect rules which exist in the Prolog file as out-of-date state. Thirdly, using Prolog querying, we can detect type 1 conflicts by solving a goal. Considering the detection of this type of feature interaction, we need to extract rules from SWRL and parts of OWL-S files automatically, but also require some common knowledge rules which will have been added by experts to reason about conflicts. If a conflict is detected, a solution will be found to resolve this type of feature interaction; if there is no conflict, we will update the Prolog file by merging the query Prolog file with the composite service execution context.

IV. PROTOTYPE IMPLEMENTATION AND CASE STUDY

In this section we present the implementation of our method and the case study will be given.

A. The implementation

We have implemented a prototype system as proof of concept. To focus on the key issue, we assume that we can obtain the OWL-S document with SWRL of the services, and that we can intercept the request and response SOAP messages of these services during their execution. The prototype is written in Java with Eclipse development environment, and uses Prolog (through the SWI-prolog plugin) for reasoning.

B. The Real world Commercial Scenario

We will illustrate our method through a concrete e-commerce example: online shopping. The scenario involves two roles: (1) A Customer who wants to buy products online and provides the basic information about products and some additional personal information. (2) Service providers providing services for online shopping in five categories, which are sales platform, shops, banks, payment intermediation and express logistics. The sales platform

services manage the deal between shop and customer, keeping the state of the transaction and support credit assessment. Shop services provide all kinds of products and make a deal with customers. Banks and payment intermediation services all are payment organizations, with payment intermediation services managing and coordinating payments among multilateral participants. Express logistics services provide delivery.

The customers' and service providers' operations can be controlled and coordinated by an intelligent agent, which implements services composition and services execution. Supposing that a customer wants to buy a digital camera online and has several requirements which include the brand and model (Canon IXUS85), the price (lower than 1400 ¥), and the place of delivery (economic development zone of Qingdao). Many feasible plans can be found through service composition based on these conditions and the available services. An example of services composition plans is shown in Fig. 2. There are six plans satisfying the customer's requirements and the second plan marked by bold lines is optimal and hence the default execution plan.

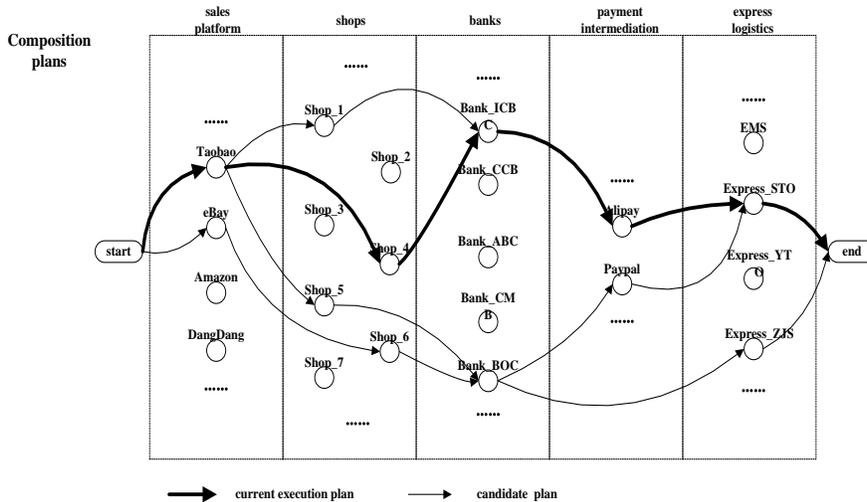


Fig. 2. Example services compositions for the case study

In this example, the two types of feature interaction can occur in the following instantiations: there is insufficient money to get the last service Express Logistics Service and there is a conflict of customer's personal address information that is available in the Banking Service becoming available to the Express Logistics Service. These interactions belong to type 1 and type 2 respectively. Our prototype detects both interactions based on rules extracted from the OWL-S file and the SOAP request message enhanced with some common knowledge. One step of feature interaction detection process is shown in Fig. 3. And with the implementation of Java and Prolog programming language, the two screen-shots (Fig 4. and Fig 5.) present a snapshot of the detection of the type 2 feature interaction. In particular these shots show the execution of the service chain (at the top) with the incoming and outgoing data (in the center part).

Most crucially, the encircled statement at the bottom shows whether a conflict has been detected or not during the execution of the last step.

V. RELATED WORK

Methods to address the feature interaction problem include offline method and online methods. Offline methods are applied during design time or for web services composition time of services, online method are applied while the features or services are being executed. Offline methods typically depend on the internal service logic (modeled at some level of abstraction). Online methods either use negotiation or feature interaction managers (FIM). Negotiation based approaches regard the components of the networks (user, terminal and value-added service, etc.)

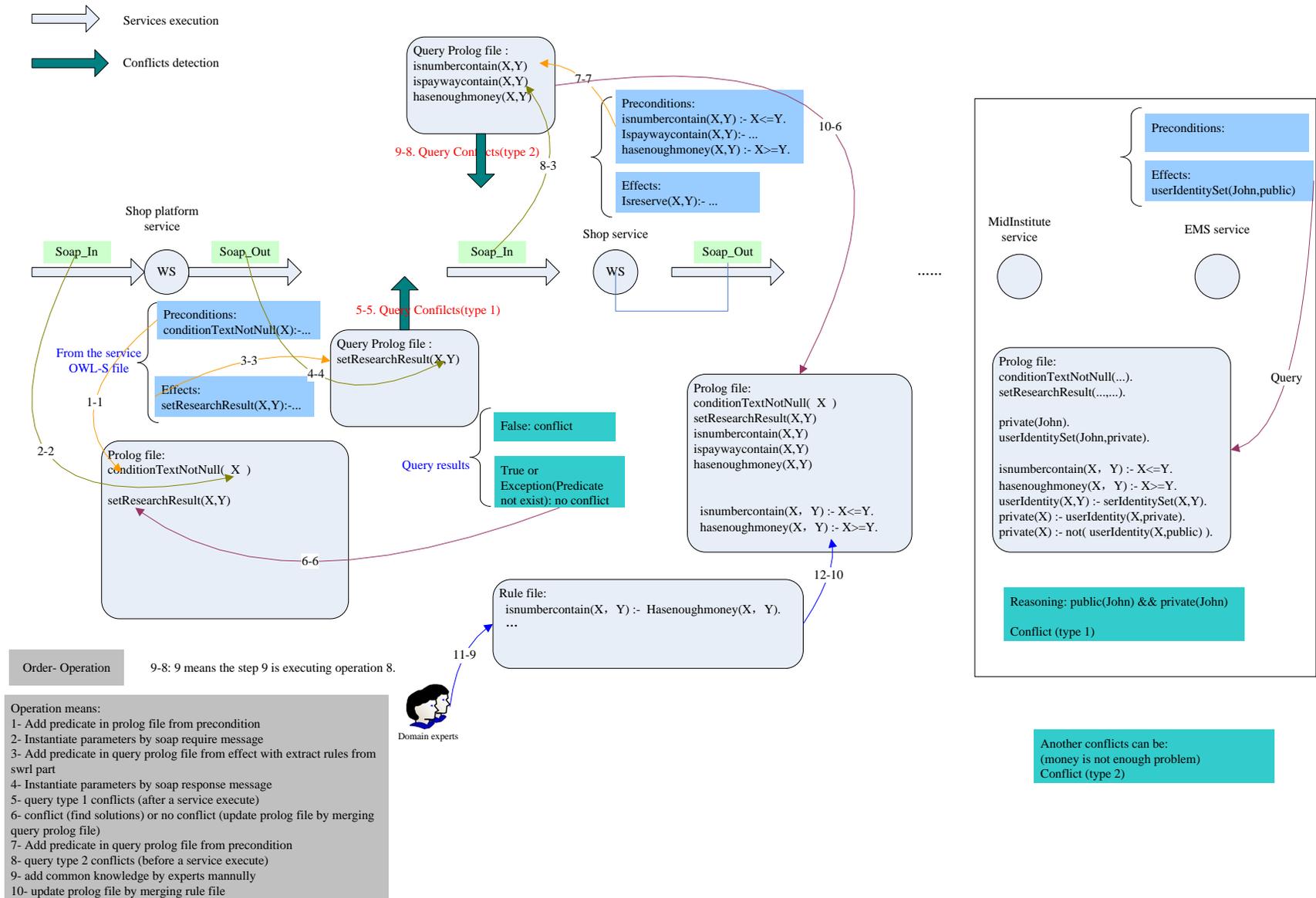


Fig.3. The Snapshot of feature interaction process with the real world scenario

as different intelligent entities and detect and resolve FI by exchanging intentions of those entities. The FIM method adds a FIM into the network to detect and resolve FI. A more detailed overview of FI methods can be found in [3].

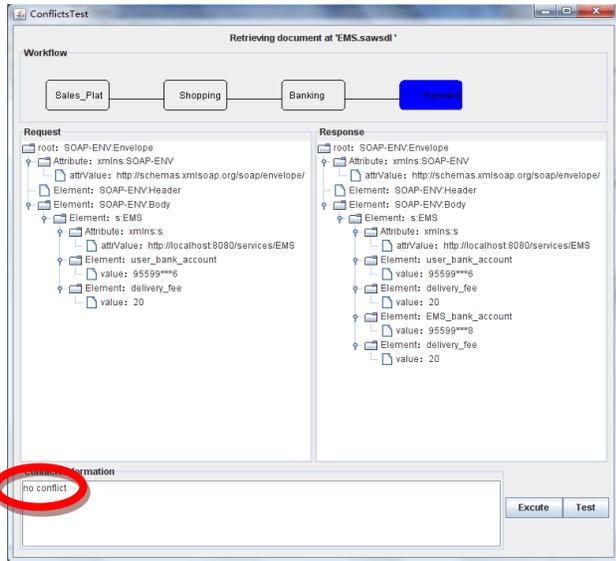


Fig. 4. The 1st picture of Series of snapshot of detection

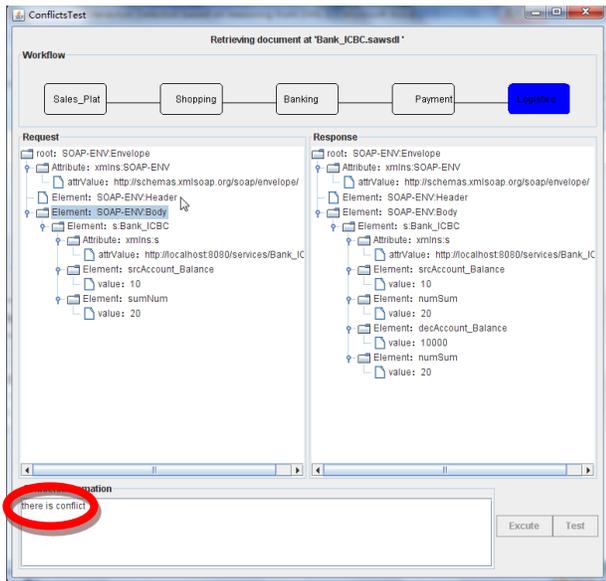


Fig. 5. The 2nd picture of Series of snapshot of detection

In the existing work on feature interaction in Web services some investigations have been conducted into offline methods and have yielded some results. Weiss et al. [6] presents a goal-oriented approach for detecting feature interactions among Web services. The authors also distinguish explicitly between functional and non-functional feature interactions. In [7] this is extended with emphasis on the classification of feature interactions. Moreover, they analyze different types of potential WSFI in a scenario.

Turner [8] extends the feature notation CRESS to graphically and formally describe Web services and service

composition. He discusses WSFI detection using CRESS and a scenario notation called MUSTARD.

Zhang et al. [9] propose a Petri net-based method to detect race conditions, which can be seen as one type of functional feature interactions. Moreover, in [4] they propose a multi-layer WSFI detection system.

Such offline methods require insight into the internal service logic, details of which might not be publically available; furthermore they require knowledge of all available services and cannot consider interactions that occur because of run-time data. We have shown that some types of interaction cannot be caught by offline methods. Moreover, in the Web service area the number of available services is very large, growing rapidly and services are offered by a large number of providers (a very open market), which does not lend itself to offline methods.

There is much research about the verification of web services properties through runtime monitoring [16-22]. Most of them [17-21] focus on the verification of properties of web services such as safety and liveness. While others [22-23] focus on monitoring exceptions at runtime. They define the possible erroneous states. In this paper, we focus on the conflicts of the side effects among the different web services used together in one business process and leading to emergent behavior despite each service running to full users' satisfaction in isolation.

In our previous work, we have considered the web service feature interaction detection based on situation calculus [15]. In this paper, we have proposed a runtime feature interaction detection method, ESTRIPS, which is an extended STRIPS method using reasoning with OWL-S and SWRL to detect FI in the Web services area to address above mentioned issues. Our method analyses the semantics of the interacting messages rather than utilizing the internal information of service logic in atomic services. The necessary details are obtained from OWL-S service descriptions and observed SOAP messages.

VI. DISCUSSION

An effective method for detecting Web service feature interaction is capable of dynamically detecting all kinds of specific known and unknown feature interactions, in a uniform manner. We presented a logical reasoning based detection method. The method overcomes the drawbacks of static detection methods mostly employed in current research and the limitation of classification-based approaches [6] which only allow for detecting interactions that are known a-priori. Our approach has the following beneficial properties:

1) The method presented is a runtime method for WSFI detection. Being a runtime method it has several advantages, one of which is that it works in an environment where new services might arrive and where there is no real potential for statically checking all possible combinations. So, this method will also work if the executed services in the workflow are dynamically identified and bound to. The actual service execution data is being used in the expression of service states.

2) The presented method is especially effective for feature interactions based on instance data of the effects,

which could include interactions related to security and privacy concerns. Such feature interactions cannot be detected by static methods as the occurrence of the interaction depends on instance data. As our method detects interactions by finding inconsistencies in the service state, data sensitive interactions can easily be detected as long as the service profile specifies the preconditions and effects of an individual service correctly.

3) The method avoids full exploration of large state spaces as it only considers services that are actually invoked together rather than all possible combinations of services and furthermore only looks at inconsistency of the service state. In that way, independent of the number of atomic services involved in the service composition, we only need to consider two states: one state before an atomic service is being executed and the state after that execution. The respective state pools might contain a large number of terms for each instance, but that data would need to be considered anyhow; however the state pools are renewed after each detection step, meaning that the information considered is local to the services of current interest.

As far as feature interaction is concerned there is a general perception that approaches in feature interaction attempt to statically determine the absence of a feature interaction. However, as shown in [3], the field of feature interaction research is quite wide and there exist run-time approaches that attempt to deal with the problem by detecting interactions and resolving them during system execution time. These approaches have inspired this work, as they are particularly adept at dealing with large numbers of services from different providers that might encounter each other for the first time when the system is running.

VII. CONCLUSION AND FUTURE WORK

With the rapid development of Web services and growing use of composite services Web service feature interaction will become a growing obstacle. While some researchers have started to address feature interaction in the web services domain, results are still very limited. By using the semantics of Web services and inspiration from logical reasoning, we proposed a novel framework and method to detect and allow for resolution of feature interactions in web services at execution time.

In future work, we intend to investigate how to decentralize the detection system. We also will test our system against more complex case studies to better evaluate efficiency and accuracy of the method. As this paper focused mostly on the feasibility of the approach in terms of detecting interactions, we are planning a more detailed evaluation of performance – clearly an important consideration for a run-time approach.

REFERENCES

- [1] Blair, L., Blair, G., Pang, J., and Efstratiou, C. Feature Interaction Outside a Telecom Domain. In: Proceedings of the Workshop on Feature Interactions in Composed Systems (ECOOP'2001) 15-20, 2001.
- [2] Amyot, D. and Logrippo, L. Guest Editorial: Directions in Feature Interaction Research. In: Computer Networks, Special Issue on Feature Interactions in Emerging Application Domains 45 5, 563-567, 2004.
- [3] Calder, M., Kolberg, M., Magill, E.H. and Reiff-Marganiec, S. Feature interaction: a critical review and considered forecast. *International Journal of Computer Networks*. 41: pp. 115-141, 2003.
- [4] Zhang, J., Yang, F., and Su, S. Detecting the Web Services Feature Interactions. In K. Aberer et al. (Eds.): WISE 2006, LNCS 4255, pp. 169-174, 2006.
- [5] Magill, E.H.. Feature Interactions: Old Hat or Deadly New Menace? In: K. Turner, E. Magill and D. Marples (Eds.), *Service Provision: Technologies for Next Generation Communications*, Wiley, pp. 235-252, 2004.
- [6] Weiss, M. and Esfandiari, B. On Feature Interactions among Web Services. In: *Proceedings of the International Conference on Web Services (ICWS)*, IEEE, 2004.
- [7] Weiss, M., Esfandiari, B., and Luo, Y. Towards a Classification of Web Service Feature Interactions, In: *Proceedings of the Third International Conference on Service Oriented Computing (ICSOC05)*, Amsterdam, Netherlands , 2005.
- [8] Turner, K.J. Formalising Web Services. In: *Proceeding of Formal Techniques for Networked and Distributed Systems (FORTE XVIII)*, LNCS 3731, 473-488, 2005.
- [9] Zhang, J., Su, S., and Yang, F. Detecting Race Conditions in Web Services, In *Proceedings of the International Conference on Internet and Web Applications and Services (ICIW'06)*, 2006.
- [10] Zhang, J., Yang, F., Shuang, K., and Su, S. Immune-Inspired Online Method for Service Interactions Detection. In Jan van Leeuwen et al. (Eds.): *SOFSEM 2007*, LNCS 4362, pp. 808-818, 2007.
- [11] OWL-S: Semantic Markup for Web Services, <http://www.w3.org/Submission/OWL-S/>.
- [12] Horrocks, I., Patel-Schneider, P.F., Bechhofer, S. and Tsarkov, D.: OWL rules: A proposal and prototype implementation. *J. of Web Semantics* 3 (2005), pp.23-40.
- [13] Redavid, D., Iannone, L., Payne, T. and Semeraro, G. OWL-S Atomic Services Composition with SWRL Rules. *Lecture Notes in Computer Science: Foundations of Intelligent Systems*, vol. 4994, pp. 605-611, 2008.
- [14] MINDSWAP: Maryland Information and Network Dynamics Lab Semantic Web Agents Project, OWL-S API. <http://www.mindswap.org/2004/owl-s/api/>.
- [15] Xu, J., Yu W., Chen K., Reiff-Marganiec, S., Web Service Feature Interaction Detection based on Situation Calculus, In: *Service' 2010*.
- [16] Turner, K.J., Reiff-Marganiec, S., Blair, L., Pang, J., Gray, T., Perry, P. and Ireland, J. Policy Support for Call Control. *Computer Standards and Interfaces*, vol 28/6 pp 635-649, 2006.
- [17] Carlo Ghezzi, Sam Guinea, Run-Time Monitoring in Service-Oriented Architectures. In: *Test and Analysis of Web Services 2007* , pp. 237-264.
- [18] Luciano Baresi, Domenico Bianculli, Sam Guinea, Paola Spoletini, Keep It Small, Keep It Real: Efficient Run-Time Verification of Web Service Compositions. In: *FMOODS/FORTE 2009*, pp.26-40.
- [19] Luciano Baresi, Sam Guinea, Marco Piistore, Michele Trainotti, Dynamo+Astro: An Integrated Approach for BPEL Monitoring. *ICWS 2009*, pp. 230-237.
- [20] Luciano Baresi, Sam Guinea, Raman Kazhamiakin, Marco Pistore, An Integrated Approach for the Run-Time Monitoring of BPEL Orchestrations. *ServiceWare 2008*, pp.1-12.
- [21] Luciano Baresi, Sam Guinea, Pierluigi Plebani, Policies and Aspects for the Supervision of BPEL Processes. *CAiSE 2007* , pp.340-354.
- [22] Katia Sycara, Roman Vaculin, Process Mediation, Execution Monitoring and Recovery for Semantic Web Services. *IEEE Data Engineering Bulletin* 31(3), pp.13-17, 2008.
- [23] Roman Vaculin, Katia Sycara, Specify and Monitoring Composite Events for Semantic Web Services, in *Web Services, 2007. ECOWS '07. Fifth European Conference on*. 2007.