Service Substitution Analysis in Protocols Evolution Context

Ali Khebizi[1], Hassina Seridi-Bouchelaghem[2], Imed Chemakhi[3], and Hychem Bekakria[3]

[1] LabStic Laboratory, 08 May 45 University, Guelma -Algeria-
ali.khebizi@gmail.com
[2] LABGED Laboratory, University Badji Mokhtar Annaba, Po-Box 12, 23000, Algeria seridi@labged.net
[3] Computer science Institute, 08 May 45 University, Guelma -Algeria-
Chemakhi.imed@gmail.com,hychem.bekakria@gmail.com

Abstract. As Web services become the dominant technology for integrating distributed information systems, enterprises are more interested by these environments. However, enterprises socio-economic environments are more and more subject to changes which impact directly business processes published as Web services. In parallel, if at change time some instances are running, the business process evolution will impact the equivalence and substitution classes of the actual service. In this paper, we present an equivalence and substitution analysis in dynamic evolution context. We suggest an approach to identify residual services that can substitute a modified one, where ongoing instances are pending. Our analysis is based on protocol schema matching and on real execution traces. The proposed approach has been implemented in a software tool which provides some useful functionalities for protocol managers.

Keywords: Service protocol, Protocol equivalence, Protocol substitution, Dynamic evolution, Execution path, Execution trace.

1 Introduction

Web services are the new generation of distributed software components. They generate a lot of enthusiasm among different socio-economic operators's which favourite these environments to deploy applications at a large scale. Standardization is a key concept, so actors uses standards like WSDL [1], UDDI [2] and SOAP [3] to publish, discover, invoke and compose distributed software. In this context, intra and inter enterprises applications integration is more flexible, easy and transparent. Moreover, integration process is accelerated among internet stakeholders.

In Web services technology, two elements are fundamental for providing a high interactivity level between service providers and service requesters. The first one is service interface, described via the standard WSDL. The second element

is service protocol (Business Protocol), which describes the provider's business process logic. A Business process consists of a group of business activities undertaken by one or more organizations in pursuit of some particular goal [4],[5]. For example, booking flight tickets and B2B transactions. In addition, Business process specifies the service external behaviour by providing constraints on operations order, temporal constraints [6] and transactional constraints [7], in order to promote correct conversations, as service operations can't be invoked in an aleatory order. However, if a service protocol is published in the Web (its interface and its protocol), at a moment during its life cycle, it can be invoked by some clients. Furthermore, in large public applications (e-commerce, e-government, electronic library, ...), thousand of clients are invoking the service at the same time and every one has reached a particular execution level. In parallel, as enterprises are open systems, changes are permanent and inevitable. Consequently, business processes may evolve to adapt to environment changes that affect real word. In this case, related service protocols must be updated, otherwise services execution may produce incoherences when they are invoked. This context is called **dynamic protocol evolution.**

In dynamic protocol evolution, the evolved service protocol may not be able to satisfy initial customer requirements. Furthermore, some services may fails and clients must find new services that can replace actual one. Services substitution analysis deal with checking if two services satisfy the same functionalities; if they support the same conversation messages [5]. This concept is very useful in case of service failure, in order to search an other one to replace it. In some cases, this analysis can serve to search and locate a new service with the same functionalities but with a higher quality of service (Qos). It can also be used to test whether a new proposed version, that expresses evolution or maintenance requirements, is yet, equivalent to an obsolete one and for finding new services that can support conversations required by standards like ebXML [8], Rosetanet [9] and xCBL [10]. Service evolution is expressed through the creation and decommission of its different versions during its lifetime. [11]

Service protocol update induces challenges for filtering which services, already identified and compared to old version, remain equivalents or can replace the evolved one. The major constraint is related to active instances that have already executed some operations based on old version. In this context, we must deal with historical executions in substitution analysis process.

In this paper we are interested to dynamic evolution and we focus on change impact analysis on service protocol substitution. A set of methods are exposed to check if new service version, can be yet substituted by the hole (or partially) set of services that were discovered corresponding to the obsolete version.

The remainder of the paper is structured as follow. We start by describing the problem and exposing our motivations, in section 2. In section 3, we propose our formal approach and algorithms for managing substitution aspects in dynamic evolution context. Section 4, describes system architecture and software tool implementation. We expose related works in section 5 and conclude with a summary and directions for future works in section 6.

2 Problem and Motivations

Every organisation (enterprises, administrations, banks, ...etc.) is an open system which is, eventually, impacted by environment changes. In order to survive, organization must adapt there business processes. Today's organizations information systems -reflecting business processes- are exposed on the Web as services (interfaces and protocols) and every business process changes induce, immediately, these two descriptions update. The challenge in dynamic protocol evolution context is to identify, among the set of already identified class substitution services, the subset of those that can, yet, replace an actual service after its specification changes, with respect to past interactions. Addressing service protocols substitution analysis, after protocol evolution, responds to the following motivations:

1. Ensuring service execution continuation for active instances.
2. Ensuring correct interactions between customers and providers by specifying the new service substitution class.
3. In dynamic environments, like Web services, transactions are long duration and resources consumer. It's not conceivable to restart execution from scratch because loss of work is catastrophic for customers.
4. In real time systems and critical applications (aeronautics, e-commerce, medical systems, control systems, manufacture,...), brutal service stop is catastrophic for organizations. It is imperative to treat with precision and accuracy services that can substitute an evolved or failed one.
5. In large public applications (e-libraries, e-government, e-learning...), a large number of active instances are pending, at a time. Manual management of these instances is cumbersome and an automatic support is required to ensure that only pertinent services are proposed for substitution process.

The main issue is to manage protocol substitution with respect to historical traces. Starting a new search query for locating new services, based on the new version, is expensive and in addition, returned services that can be inconsistent with old version.
To illustrate our motivations, we present in **Fig.1** a real world scenario.
In this scenario, service protocol P have some equivalent services $(P_1, P_2, \ldots, P_n)$ and other services $(P_3, P_4, \ldots P_m)$ can substitute it. However, service P has evolved to a new version P' (for different reasons). Evolution operations added a new message *cancelOrder* and removed the message *Order validated*. At evolution time, active instances (instance 1, instance 2,...), are running and have reached a particular execution level. In order to be able to substitute service P in case of problems, protocol manager want to know: **Which protocols remain in conformance with the new protocol specification and can replace it** ?

3 Analysing Substitution in Dynamic Protocol Evolution

One of the most challenging issues in dynamic protocol evolution context is to find potential protocols for substitution, where instances are running accord-

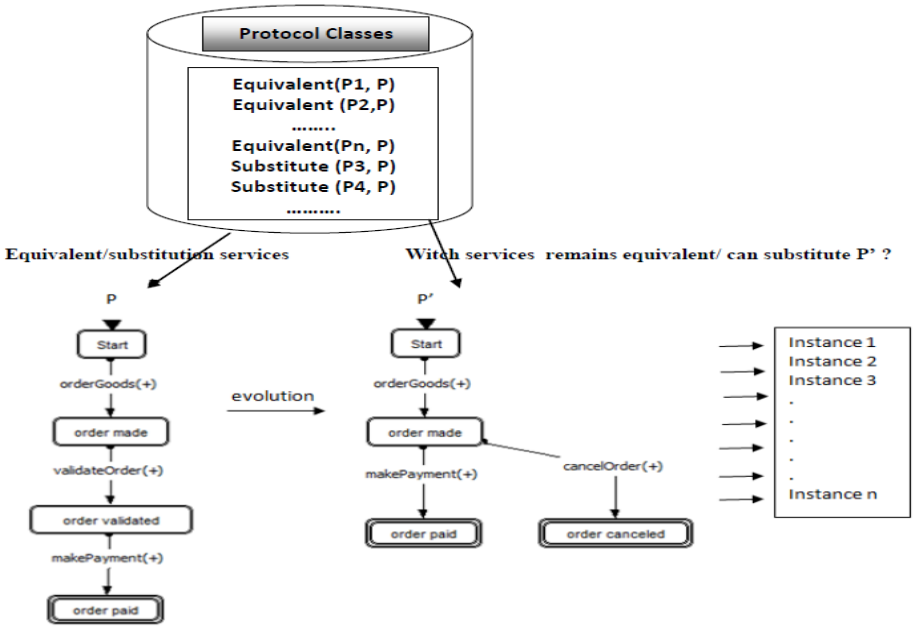

Fig. 1. After Protocol evolution of P to P' which services can substitute P' ?

ing to old protocol. To address this analysis, we introduce three fundamental concepts: **service protocol model, execution path and execution trace**.

- **A service protocol**: We use finite state machine to represent service protocols. In this model states represent different phases that a service may go through during its interaction with a requester. Transitions are triggered by messages sent by the requester to the provider or vice versa [4], [5]. A message corresponds to operation invocation or to its reply, as shown in **Fig 1**. A finite state machine is described by the tuple: $P = (S, s_0, F, M, R)$, consisting of:
  - S : A finite set of states;
  - $s_0 \in S$: is the protocol initial state;
  - F: Set of final states machine, with $F \subset S$;
  - M: Finite set of messages;
  - $R \subset (S \times S \times M)$: Transitions set. Each one involves a source state to a target state following the message receipt.
- **Execution trace**: Service behaviour traces is a finite sequence of operations $(a, b, c, d, e, ...)$. It represents events that service have invoked, from its beginning to the actual state. We note : $trace(P, i)$ to express the execution trace performed by an active instance i in a protocol P.
- **Complete Execution path**: Represents the sequence of states and messages from an initial state to a final one. We note : $expath(P)$.

3.1 Structural approach based protocol schema

Let P and P' respectively, old and new service versions, after operating changes.
$E_P = \{P_i\ (i = 1 \dots n)\}$: Is the services set **equivalent** to P. We note $Equi(P_i, P)$, the equivalence relationship between services.
Two service are equivalent if they can be used interchangeably and they provide the same functionalities in every context. Every service $Pi \in E_P$ can replace P.
$Equi(P_i, P) \Leftrightarrow \forall(i = 1 \dots n)(expath(P_i) \subset expath(P)) \wedge (expath(P) \subset expath(P_i))$.
($\wedge$ is the logic **and** operator). **(1)**
Let $S_P = \{P_j\ (j = 1 \dots m)\}$: The services set that can substitute P. We note $Subst(P_j, P)$, the substitution relationship.
A service can substitute an other one if it provides, at least, all the conversations that P supports [5] (complete execution paths).
$Subst(P_i, P) \Leftrightarrow \forall(i = 1 \dots m)(expath(P) \subset expath(P_i)$ **(2)**.
Based on this formalization we notice that if protocol P has evolved to a new version P', equations **(1)** does not remain valid. So we want to identify the protocols subset that satisfy equation **(2)**, in order to provide services that can replace the evolved protocol.
From equation **(2)**:$Subst(P_i, P') \Leftrightarrow \forall(i = 1 \dots m)\ (expath(P') \subset expath(P_i)$. **(3)** . We conclude:
Lemma 1:: If the changes related to protocol evolution are reductionist, all protocols (P_i) that can substitute P, can substitute the new reduced version P'.
Reducing Protocol description is an application of change operations including:

- Loops removal.
- Final sub-paths removal.
- Operations and messages removal.
- Complete paths removal and sub-protocols removal.

This change operation goals are motivated by procedures cancellation, reducing tasks, business processes alignment, and so one. However, when changes are additive, substitution analysis must consider the protocol difference. Protocol difference between two protocols P' and P describe the set of all complete execution paths of P' that are not common with P [5]. We note P'/P this difference.
Substitution analysis consists to examine each protocol in the class S_P, with the aim to identify possible protocols that can substitute P'. Because equation **(1)** no longer holds, we must comply with equation **(2)**.
In order to replace P', each protocol $P_i \in S_P$ must be able to replace the new requirements (the difference P'/P).
$Subst(P_i, P') \Rightarrow Subst(P_i, P'/P) \Leftrightarrow \forall(i = 1 \dots m)\ (expath(P'/P) \subset expath(P_i)$ **(4)**. We conclude:
Lemma 2: If changes are additive, protocols subset $\subset S_P$ which are containing the difference P'/P can substitute the new extended version P'. Additive changes are operations performing:

- Adding loops.
- Adding sub-paths
- Adding messages and operations .
- Adding new complete paths and sub-protocols.

3.2 Execution traces based analysis

Protocol schema based analysis is rigid and does not take into account the actual execution for active instances. Really, it's possible that a protocol $P_i \in S_P$ can't substitute an evolved one in general, but by taking into account execution traces, it can do that for specific instances. As an example, let a protocol P and its active instances $i_1, i_2, \ldots, i_n$, as mentioned in **Fig.1**. In parallel, protocol changes have added new states and messages to a particular path: *part-path*. After analysing active instances execution traces, we see that all instances have't executed this unexpected path *part-path*. In this case, even if we can't replace P' with a protocol $P_i \in S_P$, basing on protocol schema analysis, we can substitute it basing on real execution traces, because changes do not impact real instances. We notice that execution traces may inform protocol managers on how to proceed with substitution analysis. We propose two substitution analysis based execution traces: **Historical execution paths and state execution paths**.

Historical execution paths substitution analysis:
Let *histpath* a protocol p historical execution path executed by an active instance i, during its execution, instance i has invoked an operations sequence : $a, b, c, d, e, \ldots$. And, let *futurpath*: future paths not yet executed by this instance. If P' is the new version of P, after changes and S_P is the protocol set that can substitute P. We are interested by filtering instances that are not concerned with changes. We consider protocol changes as the difference between P' and $P : P'/P$. In this situation, if protocol $P_i \in S_P$ can't substitute P', contrarily, it can substitute it for the instances subset that have not executed this difference. We note: $Subst(P_i, P')/Occur_j$: The substitution of P' by P_i for occurrence j. $Subst(P_i, P')/Occur_j (i = 1 \ldots n), j = 1 \ldots m) is$ is possible if :
$(histpath(occur_j) \notin allpaths(P'/P)$.**(5)**,where $Allpaths(P'/P)$ is the hole possible paths set generated by protocol difference P'/P. This means that, substitution is possible if actual instance i had executed an old path not affected by changes.

State Based Substitution Analysis:
Historical execution path analysis is more general and based on the hole historical execution paths. Although, protocols $P' \in S_P$ can't replace P in the general case, substitution is possible for some states. We need to compute which states are not affected by changes. Substitution analysis must deal with this kind of traces by selecting protocol services that substitute active service by considering actual state and future execution path.
As an example, consider the execution path from **Fig. 1**: If a subset of actual instances are in the state: *Order made*, so their execution trace are : *begin.order made*. A service $P_i \in S_P$ can substitute P' if it can replace it from the actual state and future execution paths. We don't consider past execution paths because changes occurs after the state *(Order made)*. We formalize this analysis as follows:

Let *futurpaths* the future execution path set of an active instance (all future paths), and s the actual instance i state.
$Subst(P_i, P')/state(s) : (i = 1 \dots n)$ if :
$(futurpaths(state(s)) \subset allpaths(P'/P)) \wedge (trace(P, i) \in histpath(Pi))$.**(6)**

3.3 Algorithms

We present here **Substitution-schema-based** algorithm to calculate schema protocol based substitution analysis presented in section 3.1.
Algorithm 1: Substitution-schema-based
Input: $P = (S, s_0, F, M, R), P' = (S', s'_0, F', M', R'), P_i = (S_i, s_{0i}, F_i, M_i, R_i)$.
Output: Decision on Substitution.
Begin
1. $Substitution := True$
2. $Path = \phi,\ Completpath = \phi,\ Completpath' = \phi$
3. $Completpath := RecursivePaths(S, s_0, F, M, R)$
4. $Completpath' := RecursivePaths(S', s'_0, F', M', R')$
5. $For\ i = 1\ to\ n$ // n is protocol number
6. $While\ (path \in Completpath)\ Do$
7. $If\ (path \notin Completpath')\ Do$
8. $Substitution := False$
9. $break$
10. $EndIf$
11.$EndWhile$
12.$EndFor$
13.$Return\ (Substitution)$
14.End **Substitutions-schema-based**.
RecursivePaths Algorithm, computes recursively all possible paths in a protocol definition, from an initial state to a final one. (is not exposed by lack of space).

4 System Architecture and Software tool presentation

We have implemented the software performing substitution analysis in Java-Eclipse environment. Service protocols are implemented as automata and saved in XML files. The software performs some operational functions useful for protocol manager like protocol description and correctness specification. Furthermore, it allows final users performing different changes operations. The system kernel provides checking static equivalence and substitution. **Fig. 3**. Based on schema definitions or on execution traces, the system strength is dynamic analysis. This analysis allows user to select a particular protocol, operate changes and then proceed to change impact analysis on protocol substitution. The system filters the protocol database, analysis logs directory and searches for the remaining service protocols which can substitute the evolved version **Fig. 4**. We visualize, below some screen-shots of the the software tool.

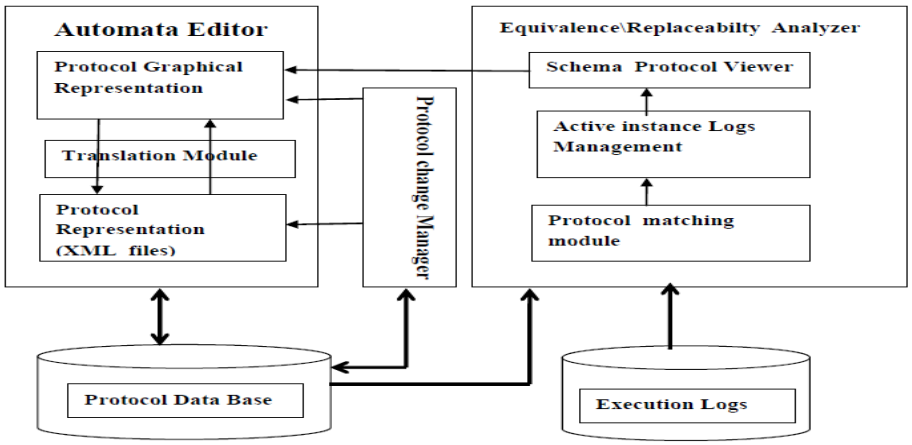

Fig. 2. System Architecture for managing dynamic substitution

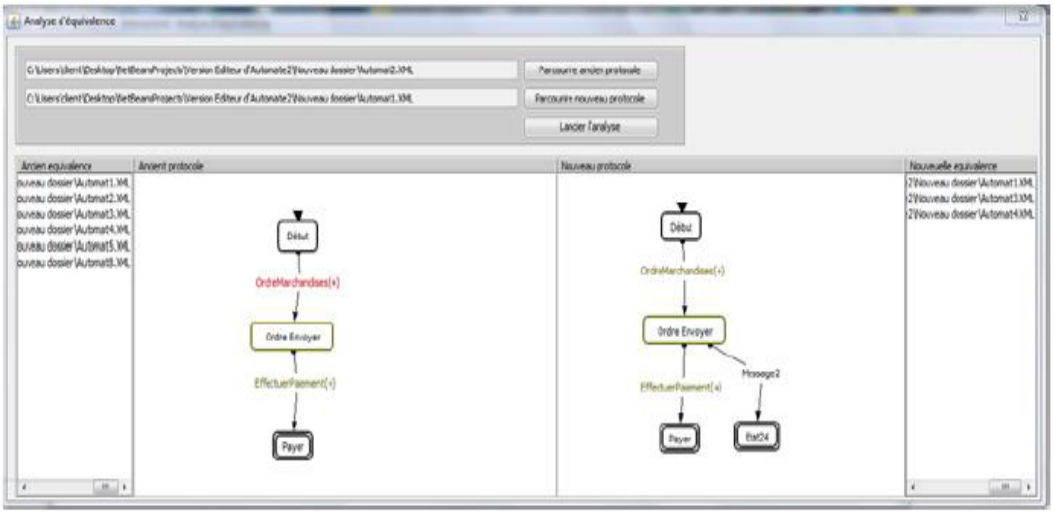

Fig. 3. Protocol specification, evolution, and static equivalence and substitution

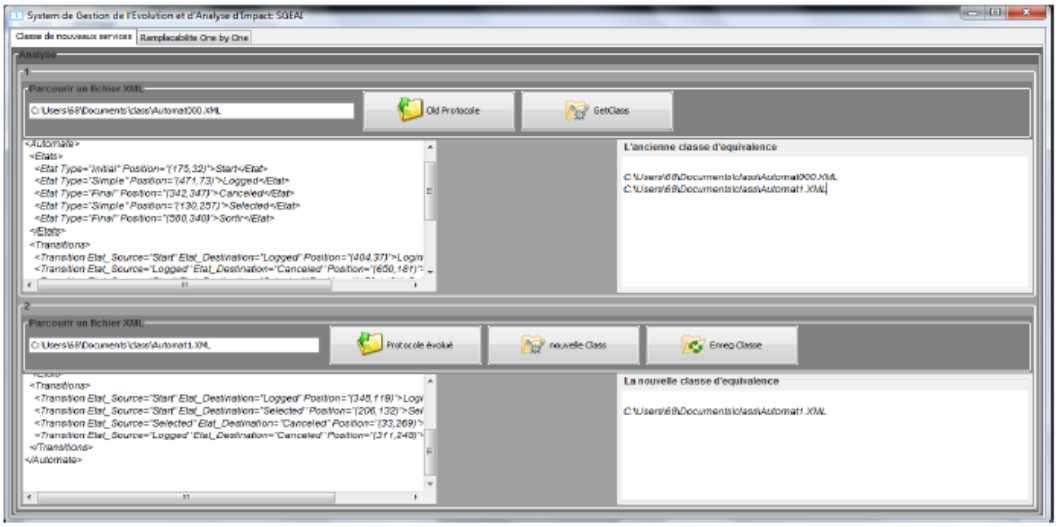

Fig. 4. Substitution analysis provide services substitution class for an evolved protocol

5 Related Work

Protocol management and analysis had benefited for a lot off contributions, from protocol schema matching to static evolution. But, dynamic protocol analysis did not receive all the interest it deserves. In [4],[5] authors presents a general framework for representing, analysing and managing Web service protocols, however this analysis is restricted to static context. In [6], protocol description is enriched with temporal constraints and the same static analysis was performed. In [12], authors had proposed some change operators and patterns specification for changes, but change impact analysis was not presented. In [13], dynamic replaceabilty analysis had been presented in therms of compatibility only. Authors studies the compatibility between old and new protocol version only. No comparison with other services was made. Our work responds in a consistent manner to the previous deficiencies.

6 Conclusion and future Work

In this article we have formalized substitution problem inherent to dynamic protocol evolution. We have proposed an approach and a software tool for providing service protocols that can, yet replace an evolved one.
As future work, we plan to address protocol substitution analysis for richer protocols descriptions, such as timed and transactional constraints automata. In addition, we aim to specify protocol changes more formally by identifying evolution patterns and by their classification with respect to induced impact on protocol substitution and compatibility.

References

1. R. Chinnici and al. Web Services description Language (WSDL) version 2.0 June 2007. http://www.w3.org/TR/wsdl20/
2. T. Bellwood and al. UDDI Version 3.0.2 UDDI Spec Technical Committee Draft, 2004. http: uddi.org/pubs/uddi-v3.htm/
3. M. Gudgin and al. SOAP version 1.2, July 2001. http://www.w3.org/TR/2001/WD-soap12-20010709/
4. B. Benatallah and al : Web Service Conversation Modeling A cornerstone for E-Business automation, IEEE Internet computing 8 (1) (2004) 46-545 WSC
5. B. Benatallah and al : Representing, Analysing and Managing Web Service Protocols. Data Knowledge Ingineering. 58 (3): 327-357, 2006.
6. J. Ponge and al: Fine-Grained Compatibility and Replaceability Analysis of Timed Web Service Protocols. ER 2007: 599-614
7. A. Khebizi: External Behavior Modeling Enrichment of Web Services by Transactional Constraints, ICSOC PhD Symposium, December 2008. http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-421/paper12.pdf
8. Technical Architecture Specification v1.0.4 February 2001 http://ebxml.org/specs/ebTA.pdf.
9. Rosetanet; http://www.rosettanet.org/.
10. xCBL; http://www.xcbl.org/.
11. Gustavo Alonso, Fabio Casati, Hurumi Kuno,Vijay Machiraju : Web services concepts Architectures and applications, Edition Springer Verlag Berlin 2004.
12. Barbara Weber and al : Change Patterns and Change Support Features - Enhancing Flexibility in Process-Aware Information Systems , 2008
13. Ryu, S. H. and al, 2008. Supporting the dynamic evolution of Web service protocols in service-oriented architectures. ACM Trans. Web 2, 2, Article 13, 46 pages. DOI = 10.1145/1346237.1346241 http://doi.acm.org/10.1145/1346237.1346241.