Extracting Behavioral Models from Executions Data in Web Services Environments

Ali Khebizi, Department of Computer Science, LabSTIC Laboratory, 8 May 1945 University P.O. Box 401, 24000 Guelma -Algeria- Email: khebizi.ali@univ-guelma.dz, ali.khebizi@gmail.com
Hassina Seridi-Bouchelaghem, LABGED University Badji Mokhtar Annaba -Algeria- Email: seridi@labged.net

Abstract*—We investigate the problem of extracting web services' protocols basing on execution log events, *i.e., how to reconstruct protocols' specifications expressing the observed behaviors from service's execution traces ?

We propose a comprehensive logic-based approach to extract business protocol models expressing the external behavior of deployed web services. Recorded execution traces are transcribed to a facts base and a set of structure inference patterns is translated to a corresponding production rules base. The conceived knowledge base is explored by a reasoning engine to infer the various elements describing the finite state machine representing the target service protocol and thus, the behavior model contained in the execution traces log files is extracted.

***Index Terms*—Web services, Service protocols, Protocol mining, Execution traces, Inference patterns, Logic programming.**

I. Introduction

Over the last few years web services are becoming the dominant technology for integrating distributed and heterogeneous information systems.

In the web service ecosystem, two elements are fundamental for providing a high interactivity level between service providers and service requesters. The first element is the service interface described via the **WSDL** standard which expresses the service localization and the allowed operations with their signature. The second one is the service protocol (*business protocol*) which reflects the provider's business process logic. A Service protocol is an abstract tool which is used to describe the service external behavior by handling conditions that govern the operations invocation, such as order and temporal constraints [**?**], [**?**].

Web services research literature has largely highlighted the usefulness of specifying service protocols and various models are proposed for their representation [**?**], [**?**]. Generally, service protocols are used to check the compatibility of customer's protocols with the published ones and to verify their conformance with other standardized processes. Also, service protocols are inevitable during the services' composition process. Thus, they constitute a cornerstone for the web services technology during the entire life-cycle: *i.e., design, enactment, management and analysis.*

Despite the importance of service protocols, various reasons can lead to their unavailability, such as: rapid service deployment, migrating legacy systems, automatic generation of the *WSDL file*...On the other hand, due to frequent changes in enterprises' environments, the current version of a service protocol can be obsolete and, consequently, it partially reflects the business logic supported by the considered business process. In such contexts, the service protocol must be reconstructed by exploring behavioral data contained in the executions logs. After that, the discovered model is improved and advertised in the web services' adequate registries.

The problem of reconstructing business protocols basing on execution traces is well known and it raises several difficulties of different natures. Most of the related issues have been extensively investigated in the research literature and several works have addressed different facets of the problem [**?**], [**?**], [**?**], [**?**], [**?**], [**?**], [**?**].

In this paper, a fundamental paradigm shift is suggested to extract service protocols from execution traces. The salient feature of the proposed work lies in a framework based on a declarative approach to support service providers in performing a fine-grained protocols reconstruction process, by customizing a set of high-level abstractions. In the proposed approach, execution traces are transcribed to a facts base which is formalized in the first-order logic predicates and a set of **structures' inference patterns** is suggested to specify the rules base that allows inferring the various elements of the target business protocol to be constructed.

Paper organization: We start by discussing related works in section 2. Section 3 presents the different facets of the proposed logic-based approach. In section 4, the system architecture is illustrated and the implementation and the experimentation of a prototype based on the use of Prolog is exposed. Finally, conclusion and potential directions for future works are drawn at section 5.

II. Related Works

Recent literature is very rich in approaches that have addressed the various challenges related to the issue of business protocol discovery. Hereafter, we discuss different works that have coped with the problem from different perspectives.

- In [**?**], [**?**], [**?**], an approach based on establishing a causality relation in the set of execution traces is suggested in order to discover the corresponding process or workflow models. The proposed α-algorithm [**?**], [**?**] uses a rules set to specify relations between activities and a Petri net, with special properties (*workflow nets*), is extracted from event logs. As the α-algorithm can't deal with short loops of length one and two, it was enhanced to α^{+}, α^{++} [**?**]. But, these last ones have residual problems with complex control-flows.

- The work presented in [**?**] proposes a formal framework to discover implicit timed transitions of conversation protocols.

The concept of *implicit transitions* was formalized to express activities that can be triggered under time constraints. In this approach, the working hypotheses are very restrictive and many conditions are imposed to model the associated constraints (*complete traces, no timed transitions leading to final states, no noise in logs, ...*).

- In [?], the authors introduce an heuristics driven process mining algorithm to discover the main behavior registered in an event log. The frequency of activities is used as a base to construct a dependency graph basing on a metric which indicates the degree of dependence between two events. This approach is applicable only to specific data having no too many different events.

- In [?], [?], the authors propose efficient algorithms to deal with the problem of event correlation in service-based processes and they characterize the set of events in service logs belonging to the same instance of a process. Such an approach is complementary to our work and it could be deployed during preliminary steps when the execution traces are not characterized by an unique identifier.

- Other significant algorithms for business processes discovery have been proposed recently (fuzzy [?], genetic [?]). In [?], a technique is suggested to evaluate these algorithms efficiently, and thus, to allow business managers selecting the appropriate algorithm that is most suitable for a given data-set.

III. Approach for Extracting Service Protocols from Execution Traces

We present below the different facets of our logic-based approach and we expose the underlying models. *(i)* First, we introduce the explicit choices on formal models used for representing service protocols and execution traces. *(ii)* The components of the used knowledge base are addressed at a conceptual level by formalizing execution traces and inference patterns. *(iii)* A reasoning engine is deployed to infer the target service protocol from the conceived knowledge base. In what follows, these steps are deeply discussed and illustrated.

A. Modeling service protocols and execution traces

We introduce below the formal models manipulated throughout the approach for specifying service protocols and execution traces.

1) The service protocol model: A service business protocol, (shortly *the service protocol*) describes the external visible behaviors of a given web service, by specifying the constraints (*e.g., ordering of messages, ...*) that customers must comply with in order to correctly interact with the service [?], [?].

While various models of different levels of expressiveness have been proposed in the literature to capture different kinds of abstractions of service protocols, in our work a basic version of the model basing on automaton is used. It describes the ordering constraints that govern the activities' execution [?], [?]. The choice of finite state machines is motivated by the important role of this formalism to represent the behavior of dynamic systems and to support formal analysis of business processes [?]. In the other hand, other existing models such UML diagrams, PetriNets and BPMN diagrams can be translated to such models by using adequate transformation techniques [?], [?].

Definition 1: A (web service) business protocol is a tuple $\mathcal{P} = (S, s_0, \mathcal{F}, \mathcal{M}, \mathcal{R})$; where:

- S is a finite set of states;
- $s_0 \in S$ is the initial state of the protocol;
- $\mathcal{F} \subseteq S$ is the set of final states;
- $\mathcal{M}$ is a finite set of abstract activities;
- $\mathcal{R} \subseteq S \times \mathcal{M} \times S$ is a transition relation. Each element $(s, m, s') \in \mathcal{R}$ represents a transition from a *source state* s to a *target one* s' upon the execution of the abstract activity m.

According to this definition, states represent the different phases that a service may go through while transitions represent activities that a service can perform to move from one step to another [?].

Example 1: A real-world example of a retirement protocol is depicted in **Fig. 1**. The protocol could be deployed as a web service by a nationwide network of social security centers for managing citizen's applications for pension benefits.

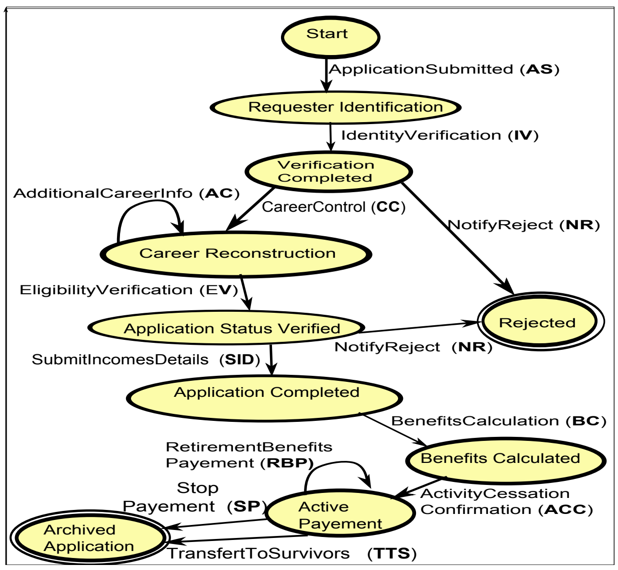
Start
ApplicationSubmitted (AS)
Requester Identification
IdentityVerification (IV)
Verification Completed
AdditionalCareerInfo (AC)
CareerControl (CC)
NotifyReject (NR)
Career Reconstruction
EligibilityVerification (EV)
Application Status Verified
Rejected
NotifyReject (NR)
SubmitIncomesDetails (SID)
Application Completed
BenefitsCalculation (BC)
RetirementBenefits Payement (RBP)
Benefits Calculated
Stop Payement (SP)
Active Payement
ActivityCessation Confirmation (ACC)
Archived Application
TransfertToSurvivors (TTS)

Fig. 1. The retirement service protocol (**Ret**)

In such a protocol, state names (e.g. Rejected) are meaningless symbols that do not affect the operational usage of the service. The transitions' labels are meaningful and correspond to the executed activities. In the sequel, the names of the activities are abbreviated as indicated in the figure (*e.g., the activity ApplicationSubmitted is abbreviated as AS*).

It should be noted that the presented service protocol is in fact an over simplified version of a real life retirement business processes but which is however, sufficient to illustrate our approach.

2) Formalizing execution traces: Each invocation of the web service by a particular customer corresponds to an execution of a separated instance of the service. This execution generates a related execution trace. We introduce bellow the concept of *execution path* which is necessary for representing execution traces of service instances.

Definition 2: An **execution path** of a protocol $\mathcal{P} = (S, s_0, \mathcal{F}, \mathcal{M}, \mathcal{R})$ is an alternating sequence $c = s_k.m_k.s_{k+1}.m_{k+1} \dots s_n.m_n.s_{n+1}$ of states and activities of $\mathcal{P}$, that (i) starts at a state s_k of $\mathcal{P}$, (ii) ends at a state s_{n+1} of $\mathcal{P}$, and (iii) is consistent with the transition relationship of $\mathcal{P}$, i.e., $(s_i, m_i, s_{i+1},) \in \mathcal{R}, \forall i \in [0, n]$.
An execution path c is called **complete** if it starts from the initial state of P (*i.e*, $s_k = s_0$) and it ends at one of the possible final states of P (*if* $s_{n+1} \in \mathcal{F}$).

According to the previous definition, an execution trace $\mathcal{T}_i$ of an instance I represents the sequence of historical activities performed by I, from the beginning of the service invocation to its current state. More formally;

Definition 3: An **execution trace** $\mathcal{T}_i$ of an instance I is a finite sequence of activities $m_0.m_1 \dots m_n$ obtained by removing the state names from the associated execution path $c = s_0.m_0.s_1 \dots s_n.m_n.s_{n+1} \in \mathcal{P}$ followed by the instance I. The execution trace is called **complete** if it is originated from a complete execution path.
We denote by $\mathcal{T}(\mathcal{P})$, the set of all execution traces $\mathcal{T}$ of a service protocol $\mathcal{P}$ and $|\mathcal{T}(\mathcal{P})|$ designates the cardinality of this set. Each element in $\mathcal{T}(\mathcal{P})$ is expressed with $\mathcal{T}_i(\mathcal{P})$, for $i = 1 \dots |\mathcal{T}(\mathcal{P})|$ and the length of $\mathcal{T}_i(\mathcal{P})$ is noted $|\mathcal{T}_i(\mathcal{P})|$, *i.e., the number of activities contained in the trace* $\mathcal{T}_i(\mathcal{P})$.

Example 2: Hereafter, we show four execution traces of different instances belonging to the retirement (**Ret**) protocol of **Fig. 1**.
•$\mathcal{T}_{22}(Ret)$=AS. IV. NR.
•$\mathcal{T}_{23}(Ret)$= AS. IV. CC.EV.
•$\mathcal{T}_{24}(Ret)$=AS. IV. CC. AC. EV. SID.
•$\mathcal{T}_{25}(Ret)$=AS. IV. CC. AC. AC. EV. NR.
Among the previous traces, $\mathcal{T}_{22}(Ret)$ and $\mathcal{T}_{25}(Ret)$ are complete, while $\mathcal{T}_{23}(Ret)$ and $\mathcal{T}_{24}(Ret)$ are incomplete ones. The length $|\mathcal{T}_{25}(Ret)|$ of the trace $\mathcal{T}_{25}$ is 7.

Before ending this section, we assume that traces are classified in a manner that it's possible to dissociate them from one service to another by using a service **Id** or a service name. From another point of view, traces may be characterized by other attributes, such as time-stamps and activities' cost. However, in the service protocol reconstruction context we focus only on the activities names which are sufficient to illustrate our approach.

B. Knowledge base specification

The knowledge base underlying our approach is articulated around, (i) a facts base containing execution traces, and (ii) a set of production rules expressing inference patterns. These two elements are addressed below.

1) Translating execution traces to a facts base: We formalize each activity of a given execution trace as a first-order logic predicate, expressed as follows.

$$A\ (Type, Order, Id, Sstate, Aname, Tstate)\ . \qquad (1)$$

Where A is a first-order predicate symbol of valence 6 and the semantics of the associated attributes are as follows.

- Type: characterizes the activity's type. The manipulated values are 1: for the first activity in the trace, 2: for the last activity and 0 corresponds to intermediate activities.
- Order: specifies the rank of the activity in the trace.
- Id: After splitting a trace to a set of separate facts, the trace's Id is integrated as an attribute for all the facts belonging to the same trace.
- Aname: designates the name of the activity in the trace.
- Sstate: is the source state $s \in \mathcal{P}$ from which the activity Aname starts.
- Tstate: is the target state $s' \in \mathcal{P}$ to which the activity $Aname$ ends.

It's forth noting that the predicate $A(Attribute)$ applied to a term of the predicate A returns the value taken by the input parameter Attribute.

According to this specification, an execution trace $\mathcal{T}_i(\mathcal{P})$ having a length $l = |\mathcal{T}_i(\mathcal{P})|$ generates a set of l separate facts. The total facts number obtained after transforming all the existing execution traces $\mathcal{T}(\mathcal{P})$ of the protocol $\mathcal{P}$ leads to a facts base, noted $\mathcal{FB}(\mathcal{P})$ having a size $|\mathcal{FB}(\mathcal{P})|$.

Example 3: According to equation (1), the traces $\mathcal{T}_{22}(Ret)$, $\mathcal{T}_{23}(Ret)$ and $\mathcal{T}_{25}(Ret)$ of example (2) are converted to the following facts base. For each fact $A_{i,j}$, i is the trace identifier and j corresponds to the attribute Order, while $S_{i,j}$ and $T_{i,j}$ are, respectively, source and target states of the activity.
•$A_{22,1}$(1,1,22,$S_{22,1}$, AS, $T_{22,1}$);
•$A_{22,2}$(0,2,22,$S_{22,2}$, IV, $T_{22,2}$);
•$A_{22,3}$(2,3,22,$S_{22,3}$, NR, $T_{22,3}$);
•$A_{23,1}$(1,1,23,$S_{23,1}$, AS, $T_{23,1}$);
•$A_{23,2}$(0,2,23,$S_{23,2}$, IV, $T_{23,2}$);
•$A_{23,3}$(0,3,23,$S_{23,3}$, CC, $T_{23,3}$);
•$A_{23,4}$(2,4,23,$S_{23,4}$, EV, $T_{23,4}$);
•$A_{25,1}$(1,1,25,$S_{25,1}$, AS, $T_{25,1}$);
•$A_{25,2}$(0,2,25,$S_{25,2}$, IV, $T_{25,2}$);
•$A_{25,3}$(0,3,25,$S_{25,3}$, CC, $T_{25,3}$);
•$A_{25,4}$(0,4,25,$S_{25,4}$, AC, $T_{25,4}$);
•$A_{25,5}$(0,5,25,$S_{25,5}$, AC, $T_{25,5}$);
•$A_{25,6}$(0,6,25,$S_{25,6}$,EV, $T_{25,6}$);
•$A_{25,7}$(2,7,25,$S_{25,7}$,NR, $T_{25,7}$);
In the previous facts base, the predicate $A_{23,3}(Aname)$ applied to the parameter $Aname$ returns the value CC.

2) Inference patterns specification and semantics: We propose a set of generic **Inference Patterns (IP)** to capture descriptive elements of the target protocol by exploring execution traces. The proposed patterns define recurrent situations occurring during protocol invocation and they are intended to combine in a single specification, both the syntactic elements associated to execution traces (*activities*) with the structural concepts manipulated at a high abstract level and expressing protocol schema *(transitions, nodes, split, join ...)*. Inference patterns are described with a pattern name, a formal specification and a textual description.

a) Pattern specification: Let $\mathcal{T}$ be a collection of traces represented as a set of facts and let $\mathcal{P}$ be the target service protocol to be extracted. A general specification of an inference pattern is as follows.

$$TargetElement \models IP(Scope, Param), \quad where: \qquad (2)$$

- $TargetElement$ constitutes the logic conclusion inferred from premises expressed through the predicate IP. This goal specifies a particular element of the protocol (*states, sequences, loops,...*) that to be discovered.

- IP is a predicate in the first-order logic which captures the semantics of the pattern type.
- $Scope$ is a constraint over the execution traces set $\mathcal{T}$, i.e., execution traces explored using this pattern.
- $Param$ is a set of optional parameters expressing values related to the protocol identifier, the activities' name or other traces attributes.

b) Pattern semantics: The inference pattern specification stipulates that a subset of execution traces satisfying the pattern's *Scope* is used in input to evaluate the predicate $IP(Scope, Param)$, while considering the set of parameters *Param*. If this predicate is evaluated to True then the goal is satisfied and a corresponding *TargetElement* is identified as a structure of the target protocol.

3) Inference patterns identification and formalization: In the following eight inference patterns are identified and formalized. The first three ones are intended to extract static components (*states*) while the five last ones concern dynamic structures (*transitions*). For each identified pattern, we give its formal specification, we explain its semantics and we illustrate its usage through an example.

a) Initial state pattern (IP1): In order to identify the initial state of a protocol $\mathcal{P}$, the set of first activities contained in the associated facts base $\mathcal{T}(\mathcal{P})$ is filtered out. The constraints governing the initial state specification are expressed by the following inference pattern.

$$s_0 \models InitialState(\mathcal{T}, \mathcal{P}). \tag{3}$$

The semantics of the pattern stipulates that the scope of the pattern is given by the subset of traces $\mathcal{T}$ of $\mathcal{P}$; i.e., $\mathcal{T}(P)$. After transforming the filtered traces to a facts base $\mathcal{FB}(\mathcal{P})$, the initial state s_0 of the protocol is discovered if the predicate $InitialState(\mathcal{T}, \mathcal{P})$ is evaluated to True. This predicate is evaluated to True if the following condition holds.

$$\exists\ i, j, \text{ such that: } A_{i,j} \in \mathcal{FB}(\mathcal{P}), \text{ and } A_{i,j}(Type) = 1.$$

As an illustration, the previous pattern is satisfied for the following three facts of the facts base of example (3).
•$A_{22,1}(1, 1, 22, S_{22,1}$, AS, $T_{22,1})$;
•$A_{23,1}(1, 1, 23, S_{23,1}$, AS, $T_{23,1})$;
•$A_{25,1}$(1,1,25,$S_{25,1}$, AS, $T_{25,1}$).
Furthermore, as the three activities' names are identical; (i.e, AS), an unique execution path starting from the state s_0 is created in the target automaton.

b) Final states pattern (IP2): The inference pattern specifying the final states $F_l \subseteq S$ of a target protocol is formalized as follows.

$$F_l \models FinalStates(\mathcal{T}, \mathcal{P}),\ with\ l \geq 1\ . \tag{4}$$

The final states of $\mathcal{P}$ are recognized by focusing on target states of last activities. Thus, a fact $A_{i,j}$ expressing a transition $(s, m, s') \in \mathcal{R}$ is ending at a final state, only if the predicate $FinalStates(\mathcal{T}, \mathcal{P})$ is evaluated to True. This predicate is satisfied if the following condition holds.

$$\exists\ i, j, \text{ such that: } A_{i,j} \in \mathcal{FB}(\mathcal{P}), \text{ and } A_{i,j}(Type) = 2.$$

Whenever a final state F_l is discovered, it is added to the already discovered final states set; *i.e.,* $\mathcal{F} = \mathcal{F} \cup\ F_l$.

As an illustration, the deployment of the pattern **IP2** of equation (4) to the facts base of example (3) produces the following subset of facts.
•$A_{22,3}(2, 3, 22, S_{22,3}$, NR, $T_{22,3})$;
•$A_{23,4}(2, 4, 23, S_{23,4}$, EV, $T_{23,4})$;
•$A_{25,7}(2, 7, 25, S_{25,7}$, NR, $T_{25,7})$.
Thus, three final states F_1, F_2 and F_3 are discovered and added to the final states set $\mathcal{F}$ of the protocol $\mathcal{P}$; i.e;, $\mathcal{F} = \mathcal{F} \cup \{\ F_1, F_2, F_3\}$.

c) Intermediate states pattern (IP3): From a structural point of view, an intermediate state is a target state for an ingoing activity and a source state for an outgoing one. The corresponding inference pattern is formalized as follows.

$$S_k \models IntermdediateState(\mathcal{T}, \mathcal{P}),\ with\ k \geq 1. \tag{5}$$

The Pattern semantic indicates that a new intermediate state S_k is discovered in the protocol $\mathcal{P}$, if the predicate $IntermediateState(\mathcal{T}, \mathcal{P})$ applied to the facts base $\mathcal{FB}(\mathcal{P})$ is evaluated to True. This predicate is True for two facts $A_{i,j}$ and $A_{i,j'} \in \mathcal{FB}(\mathcal{P})$ if their related activities are consecutive ones. More formally.

$$A_{i,j'}(Order) = A_{i,j}(Order) + 1;\ for\ i \in [1, |\mathcal{T}(\mathcal{P})|] \text{ and } j, j' \in [1, |\mathcal{T}_i(\mathcal{P})|].$$

During the intermediate states reconstruction process, the target state $T_{i,j}$ of the first activity and the source state $S_{i,j+1}$ of the consecutive one are renamed with the same state name $\mathcal{S}_k$.

As an example, when applying the previous pattern to the facts base of example (3) with the particular value of $i = 22$, two intermediate states are discovered. After the states extraction, the variables $T_{22,1}$ and $S_{22,2}$ in example (3) are renamed to $\mathcal{S}_1$. In a similar fashion the states $T_{22,2}$ and $S_{22,3}$ are renamed to $\mathcal{S}_2$. The discovered two states $\mathcal{S}_1$ and $\mathcal{S}_2$ are added to the set of states: $\mathcal{S} = \mathcal{S} \cup \{\mathcal{S}_1, \mathcal{S}_2\}$.
By taking into account the already discovered initial and final states, the improved facts associated to the trace $\mathcal{T}_{22}(Ret)$ become as follows.
•$A_{22,1}(1, 1, 22, S_0$, AS, $S_1)$;
•$A_{22,2}(0, 2, 22, S_1$, IV, $S_2)$;
•$A_{22,3}(2, 3, 22, S_2$, NR, $F_1)$.

d) Self-loop pattern (IP4): The existence of self-loop structures is manifested by execution traces that exhibit activities sequences of the form $m.m.m \ldots$. The following inference pattern allows detecting states S_j that exhibit such a behavior.

$$S_j \models Loops(\mathcal{T}, (\mathcal{P}, A)),\ with\ j \geq 0. \tag{6}$$

The self-loop inference pattern is customized with two parameters; the protocol $\mathcal{P}$ and the activity A concerned by the loop test. In the protocol specification, a state S_j holds a loop structure upon the execution of an activity A, if the predicate $Loops(\mathcal{T}, (\mathcal{P}, A))$ is evaluated to True. This predicate is True if:

$$\exists\ i, j\ (j \in [2, |\mathcal{T}_i(\mathcal{P})| - 1]), \text{ such that: } A_{i,j}, A_{i,j+1} \in \mathcal{FB}(\mathcal{P}) \text{ and } A_{i,j}(Aname) = A_{i,j+1}(Aname) = A$$

In the previous condition, the constraint $j \in [2 \ldots |\mathcal{T}_i(\mathcal{P})| - 1]$ ensures that loops can't be built on the initial state ($j \neq 1$) and final ones ($j \in |\mathcal{T}_i(\mathcal{P})| - 1$).

The exploration of the facts base of example (3) basing on the self-loop pattern of equation (6) leads to the two following facts that satisfy the predicate $Loops(\mathcal{T}, (Ret, AC))$ upon the execution of the activity AC.
•$A_{25,4}(0, 4, 25, S_{25,4}$, AC, $T_{25,4})$;
•$A_{25,5}(0, 5, 25, S_{25,5}$, AC, $T_{25,5})$.
After renaming states as follows:
$S_4 = S_{25,4} = T_{25,4} = S_{25,5} = T_{25,5}$, the two previous facts become:
•$A_{25,4}(0, 4, 25, S_4$, AC, $S_4)$,
•$A_{25,5}(0, 5, 25, S_4$, AC, $S_4)$.

e) Activities sequence pattern (IP5): A sequence structure links chronologically, at least, two activities $A_{i,j}$ and $A_{i,j'}$ by a common state $\mathcal{S}_k$. Such structures are expressed by two transitions $(s, m, \mathcal{S}_k)$ and $(\mathcal{S}_k, m', s') \in \mathcal{R}$. The following inference pattern allows extracting sequences constructs from the activities facts base.

$$(A_{i,j}, A_{i,j'}) \models Sequence(\mathcal{T}, (\mathcal{P}, S_k)). \qquad (7)$$

The semantics of the pattern is interpreted as follows. First, only traces of the input protocol $\mathcal{P}$ are handled. When exploring the corresponding facts base $\mathcal{FB}(\mathcal{P})$, for each state S_k introduced as an input parameter of the pattern, the pairs of consecutive activities $(A_{i,j}, A_{i,j'})$ are searched by locating incoming activities $A_{i,j}$ and the outgoing ones $A_{i,j'}$, while evaluating the predicate $Sequence(\mathcal{T}, (\mathcal{P}, S_k))$. For a given state S_k this predicate is evaluated to True, for all the facts $A_{i,j}$ and $A_{i,j'} \in \mathcal{FB}(\mathcal{P})$ satisfying the condition.

$$\forall\ i, j, j', \text{ such that: } A_{i,j}, A_{i,j'} \in \mathcal{FB}(\mathcal{P}); \\ A_{i,j}(Tstate) = A_{i,j'}(Sstate) = S_k\ (S_k \notin \mathcal{F})$$

By applying the pattern **IP5** to the facts sub-base presented at the end of the pattern **IP3** (*see section III-B3c*) with S_2 as a parameter, the predicate $Sequence(\mathcal{T}, (Ret, S_2))$ is evaluated to True for the two activities IV and NR *(j=2 and j'=3)*, because $A_{22,2}(Tstate) = A_{22,3}(Sstate) = S_2$. Thus, it's inferred that the state S_2 links the two activities IV and NR. Consequently, a sequence structure is discovered.

f) Join structure pattern (IP6): Join structures express the convergence of several activities to an unique state of the automaton. The following pattern is used to detect activities involving a join structure over a state S_k of $\mathcal{P}$.

$$A_{i_1,j_1}, A_{i_2,j_2}, \ldots, A_{i_n,j_m} \models Join(\mathcal{T}, (\mathcal{P}, S_k)). \qquad (8)$$

The pattern semantics indicates that a state S_k introduced as an input parameter of the pattern represents a join node for a sub-set of activities performed in execution traces $\mathcal{T}$ of a protocol $\mathcal{P}$, if the predicate $Join(\mathcal{T}, (\mathcal{P}, S_k))$ is evaluated to True. This predicate is True if the following condition holds.

$$\exists\ i_1, i_2, \ldots, i_n, j_1, j_2, \ldots, j_m \text{ , such that:} \\ A_{i_1,j_1}, A_{i_2,j_2}, \ldots, A_{i_n,j_m} \in \mathcal{FB}(\mathcal{P}) \text{ and } A_{i_1,j_1}(Tstate) = \\ A_{i_2,j_2}(Tstate), \ldots, = A_{i_n,j_m}(Tstate) = S_k$$

For instance, assume that four facts $A_{20,2}$, $A_{30,3}$, $A_{40,5}$ and $A_{50,8}$ of a purchase protocol are described according to the facts predicate of equation (1).
•$A_{20,2}(0, 2, 20, S_2$, Select, $T_2)$;
•$A_{30,3}(0, 3, 30, S_3$, Validate, $T_3)$;
•$A_{40,5}(0, 5, 40, S_5$, Purchase, $T_4)$;
•$A_{50,8}(0, 8, 50, S_8$, Payment, $T_4)$.
According to equation (8), the predicate $Join(\mathcal{T}, (Ret, T_4))$ is evaluated to True for the two last facts:
$\{A_{40,5}, A_{50,8}\}$, ($i_1 = 40, i_2 = 50$ and $j_1 = 5, j_2 = 8$) and the state T_4 constitutes a join node for the activities Purchase and Payment.

g) Split structure pattern (IP7): A split structure represents a subset of distinguished transitions which are outgoing from a same source state S_k. The specification of the inference pattern for discovering split nodes in the facts base $\mathcal{T}(\mathcal{P})$ is formalized as follows.

$$A_{i_1,j_1}, A_{i_2,j_2}, \ldots, A_{i_n,j_m} \models Split(\mathcal{T}, (\mathcal{P}, S_k)). \qquad (9)$$

The semantic carried by this pattern is interpreted as follows. A state S_k introduced as a parameter of the pattern constitutes a split node for the outgoing activities $A_{i_1,j_1}, A_{i_2,j_2}, \ldots, A_{i_n,j_m}$, if the predicate $Split(\mathcal{T}, (\mathcal{P}, S_k))$ is evaluated to True. This predicate is satisfied if the following constraint holds.

$$\exists\ i_1, i_2, \ldots, i_n, j_1, j_2, \ldots, j_m \text{ , such that:} \\ A_{i_1,j_1}, A_{i_2,j_2}, \ldots, A_{i_n,j_m} \in \mathcal{FB}(\mathcal{P}) \text{ and } A_{i_1,j_1}(Sstate) = \\ A_{i_2,j_2}(Sstate), \ldots, = A_{i_n,j_m}(Sstate) = S_k$$

As an illustration, let us consider the two following facts of example (*3*).
•$A_{22,3}(2, 3, 22, S_{22,3}$, NR, $T_{22,3})$;
•$A_{23,3}(0, 3, 23, S_{23,3}$, CC, $T_{23,3})$.
According to the intermediate state pattern **IP3**, the states $S_{22,3}$ and $S_{23,3}$ are renamed to S_3, leading to the facts:
•$A_{22,3}(2, 3, 22, S_3$, NR, $T_{22,3})$;
•$A_{23,3}(0, 3, 23, S_3$, CC, $T_{23,3})$.
The predicate $Split(\mathcal{T}, (\mathcal{P}, S_k))$ is evaluated to True for the state S_3 which constitutes a split node for the two activities NR and CC.

h) Cycles on sub-structures (IP8): Instead of looping on simple states, service instances may iterate a whole activities sequence during their executions. More formally, consider that I is an instance of a web service with its trace $\mathcal{T}_i$ and let $[A_{i,1}.A_{i,2}. \ldots .A_{i,l}]$ be an activities sequence of length l (*with* $l > 1$) which is performed by I during its interaction with the service. If the previous activities sequence $[A_{i,1}.A_{i,2}. \ldots .A_{i,l}]$ occurs more than once in the same trace, it is deduced that a cycle on a sub-structure exists in the protocol specification. The following pattern is deployed to extract potential existing cycles on sub-structures of the protocol.

$$[A_{i,j}.A_{i,j+1}. \ldots .A_{i,j+l}] \models Cycle(\mathcal{T}, \mathcal{P}). \qquad (10)$$

The semantics of the pattern is interpreted as follows. A cycle is detected for an activities sequence $[A_{i,j}.A_{i,j+1}. \ldots .A_{i,j+l}]$, if the predicate $Cycle(\mathcal{T}, \mathcal{P})$ is evaluated to True. This predicate is satisfied for the previous sequence if:

$$\exists\ i, j \text{ , such that: } A_{i,j}, A_{i,j+1}, \ldots, A_{i,j+l} \in \mathcal{FB}(\mathcal{P}) \text{ and} \\ Occurrence([A_{i,j}, A_{i,j+1}, \ldots, A_{i,j+l}]) > 1.$$

For identifying potential cycles in the target protocol, possible duplicated occurrences of the activities sequence in a trace $\mathcal{T}_i$ are examined depending on different values of the length $1 < l \leq |\mathcal{T}_i|$.

As an illustration, consider an instance $I = 100$ with its execution trace $\mathcal{T}_{100}(\mathcal{P}) = a.b.c.d.b.c.e$, having the two redundant activities b and c. According to the facts representation, (*see subsection (III-B1)*), the trace $\mathcal{T}_{100}(\mathcal{P})$ is transformed to the following subset of facts.
•$A_{100,1}(0, 1, 100, S_{100,1}$, a, $T_{100,1})$;
•$A_{100,2}(0, 2, 100, S_{100,2}$, b, $T_{100,2})$;
•$A_{100,3}(0, 3, 100, S_{100,3}$, c, $T_{100,3})$;
•$A_{100,4}(0, 4, 100, S_{100,4}$, d, $T_{100,4})$;
•$A_{100,5}(0, 5, 100, S_{100,5}$, b, $T_{100,5})$;
•$A_{100,6}(0, 6, 100, S_{100,6}$, c, $T_{100,6})$.
•$A_{100,7}(0, 7, 100, S_{100,7}$, e, $T_{100,7})$.
For the same trace $\mathcal{T}_{100}$, the activities b and c appear more then once in the facts base. Further, for the value $l = 2$, the sequence b.c is redundant in the facts base and thus, it is deduced that a sub-structure performing a cycle on the activities sequence b.c exists in the service protocol schema.

C. Exploration of the knowledge base

A multi-stages process is conducted to explore the conceived knowledge base in order to extract the structural elements of the target protocol. These steps are:

1) Once the traces database is imported from providers' information systems, a pre-processing phase is initiated. It consists in removing noises and unnecessary attributes, such as information about service quality (*QoS*) and time-stamp. Further, in order to optimize the overall efficiency of the proposed approach redundant and included traces are removed. Thus, if the traces $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_n$ are contained in a trace $\mathcal{T}_m$, then only $\mathcal{T}_m$ is retained to be exported to the cleaned traces database.
2) Cleaned execution traces are exported to an adequate database that is ready to be exploited as input of the knowledge base reasoning system.
3) Each execution trace $\mathcal{T}_i$ of length l is transcribed to l separate facts which are stored in the facts base. In the other hand, the formalized inference patterns are implemented as production rules. Facts and rules are expressed in a low level language such as *Prolog*.
4) An inference engine using backward chaining (*Prolog*) is deployed to produce the descriptive elements of the target protocol by assessing the production rules. The attributes of inferred structures are stored as XML elements expressing *state names and their types, transition names and their attributes,. . . .* Additionally, conformance checking actions are performed to avoid errors and inconsistency that can occur in the built protocol, such as *unreachable states, absence of initial and final states*.
5) For ergonomic viewing reasons, a graphical representation of the built protocol is provided to end users to allow them refining and enhancing the produced protocol in a visual manner.

IV. Implementation and Experiments

This section briefly describes a prototype, named **L**ogical **B**usiness **P**rotocol **R**econstructor (**LBPR**) which implements the proposed approach. First, the system architecture is presented then preliminary experimental results are discussed.

A. System Architecture and functionalities

Java environment integrating useful Eclipse plug-ins, such as MySQL-connector and **JPL** (Java Programming Logic) [**?**] have been used to implement the prototype **LBPR**. The **JPL** library allows incorporating Prolog in Java and it's used by Java for interacting with SWI-Prolog [**?**]. As illustrated in **Fig. 2**, the prototype **LBPR** is organized around three main components interacting with the conceived knowledge base.

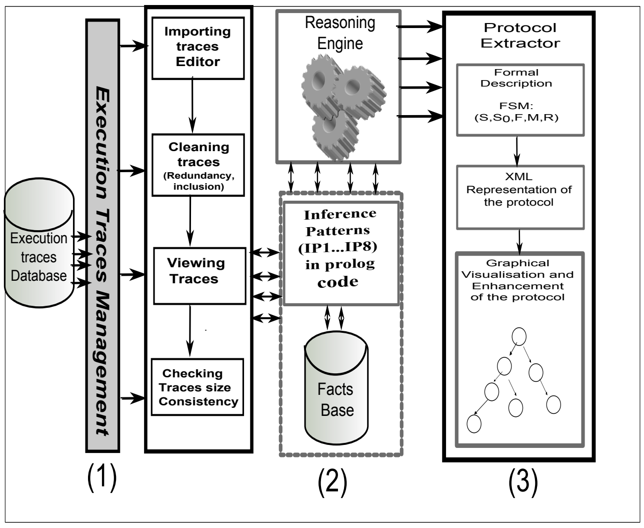
Execution traces Database
Execution Traces Management
Importing traces Editor
Cleaning traces (Redundancy, inclusion)
Viewing Traces
Checking Traces size Consistency
Reasoning Engine
Inference Patterns (IP1...IP8) in prolog code
Facts Base
Protocol Extractor
Formal Description FSM: (S,S0,F,M,R)
XML Representation of the protocol
Graphical Visualisation and Enhancement of the protocol
(1)
(2)
(3)

Fig. 2. System Arhitecture and functionalities

The first component of **LBPR** is the Execution Traces Management module which allows users connecting to the log files database in order to select and to import execution traces of the concerned web service. After that, a pre-processing step is initiated. It consists in removing redundant and included traces (module: cleaning traces). At this stage the relevance and the representativeness of imported traces are evaluated by calculating the rate ρ of the complete traces with regard to the total number of the imported traces: $\rho = C/P$; *with C= Complete traces and P = Imported traces*. This verification is ensured by the module *Checking traces size and Consistency*.

The second component of **LBPR** enables constructing the facts base in accordance to the facts predicate model of equation (**1**). Once the facts base is generated, the system performs automatically the extraction of the protocol as a sequence of chronological actions. In each step, a particular structure inference pattern is activated. The protocol extraction process is controlled through parameters settings which determine the inference pattern that should be performed, some adjustable thresholds and the levels of details for writing the reasoning engine logs. The user gets a detailed log of all mining steps expressed by native prolog traces.

The last component of **LBPR** allows different representation of the built protocol. In fact, upon patterns execution the induced goals constitute the descriptive elements of the target protocol. The discovered structures are managed in XML-format and are stored in adequate files describing the target automaton. Each XML element in the output file represents either a protocol state or a transition between states. The attributes associated to XML elements express values of properties characterizing states names, transitions names and states types, as well as the screen positions manipulated during the graphical visualization of the protocol. For ergonomic reasons, the prototype **LBPR** was consolidated by a graphical editor that allows viewing and browsing the extracted protocols.

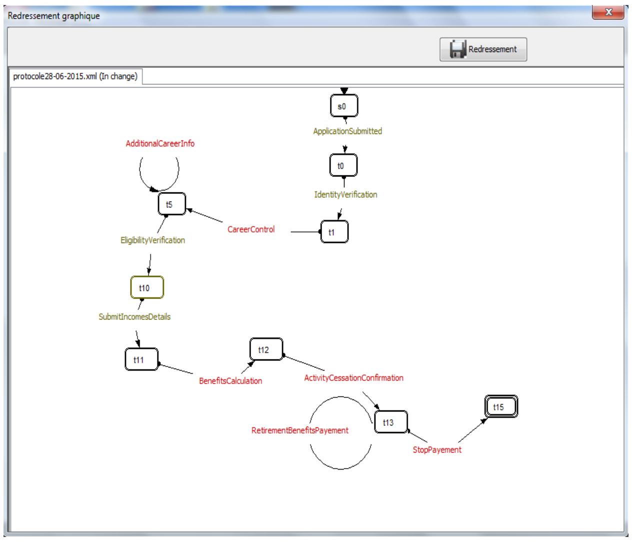

Fig. 3. Graphical viewing and browsing of the target protocol

As depicted in **Fig. 3**, a partial extracted retirement protocol is produced by **LBPR** from execution traces data-set. Furthermore, in order to improve the protocol specification and visualization, a graphical tool box provides useful functions, like moving states/activities from one position to another, adding or renaming states/activities and finally removing elements. Another useful feature of **LBPR** is the conformance-checking tool which ensures the verification of a set of correctness criteria (*initial, final and unreachable states* . . .).

B. Experiments

To evaluate the scalability and the performance of the proposed approach, we conducted experiments with the prototype **LBPR** over synthetic data-sets originating from various fields (*pension, e-commerce, booking flights*) and containing respectively: DS1=1.000, DS2=10.000, DS3=100.000 and DS4=200.000 traces.

In a first experiment, we focused on the pre-processing step by evaluating the rates of redundant and included traces, while recording the elapsed time during data cleaning. To this end, the data-set DS3 containing 100.000 traces was chosen and treated. It was observed that the rate of redundant execution traces represents 46 % of the total size of the introduced data-set. Once duplicated traces were removed, only filtered ones are submitted to an inclusion checker module which tests traces inclusion and discards the included traces for the next experimental steps. Empirical observations show that average 38 % of the total traces are included in other ones, because we are dealing with real business processes characterized by a bounded number of execution paths. After the pre-processing step, execution traces are reduced approximately to 34 % of their original size. These proportions are largely suitable for the remainder of the experiment process. In the other hand, the observation of the manipulated data shows that the average of activities is generally around 10 activities per trace. Consequently, after the pre-processing step, the size of corresponding facts in each data-set approximates, respectively, $\mathcal{BF}1$=3.400, $\mathcal{BF}2$=29.700, $\mathcal{BF}3$=272.000 and $\mathcal{BF}4$=594.000.

To examine the efficiency and the relevance of the proposed patterns, in a second experiment the time spent during the reasoning step is evaluated. Hence, when chronologically activating the various inference patterns, the time consumed by the reasoning engine to resolve Prolog clauses was calculated.

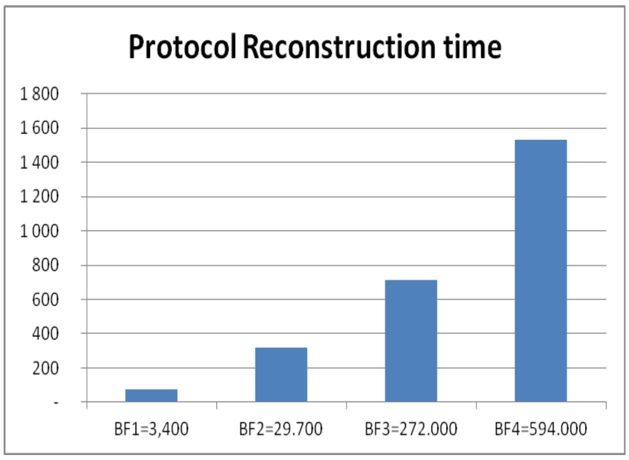

Fig. 4. Variation of the protocol extraction time

As shown in **Fig. 4**, this time grows linearly according to the size of the facts base introduced in input. As the protocol extraction process operates on cleaned data, the obtained results are very satisfactory and turn around 25 minutes (1532 seconds) for the largest data-set DS4 (*BF* $\simeq$ *594.000 facts*). These results encourage the system deployment for handling large-public applications (*e-government, e-learning, e-commerce,* . . .) which are characterized by a high size of execution data. Nevertheless, when viewing the resulting protocol as a finite state machine different display problems arise, due the aleatory attributes associated to the automaton states in the XML files. In fact, we associate to each discovered state two random coordinates (x,y) to display it on the screen. To overcome this limitation, while enhancing protocol's viewing, user's manual intervention is required. In this trend, system users can drug and drop states and transitions from one position to another. The new values of the graphical positions are captured and stored as attributes of the automaton description.

V. Conclusion and Future Work

In this work an approach based on first-order logic predicates is proposed to extract web services business protocols. Activities expressing log events are transcribed to first-order predicates and are implemented as Prolog clauses. Furthermore, inference patterns have been identified and formalized as a key concept for business protocol extraction. The discovered protocols are **complete**; *i.e., all recorded traces are covered by the extracted protocol*, and **minimal**; *i.e., future interactions can't be predicted and handled by the system.*

Beyond the benefits from logic programming paradigm (*extensibility, traceability, easy implementation and best performances…*), the main contributions of this work are: (i) a logic formal framework for extracting web services protocols is suggested. (ii) a technique based on redundant and included traces is adopted to reduce the search space. (iii) the proposed approach is declarative, configurable and customizable. (iv) the proposed approach is implemented and experimented using real-life data.

In future works, we plan to improve the output protocol viewing. As a potential direction, we project to deploy and experiment the approach on big data originating from social networks.

References