

Web Service Mining and Verification of Properties: An approach based on Event Calculus

Mohsen Rouached

▶ **To cite this version:**

Mohsen Rouached. Web Service Mining and Verification of Properties: An approach based on Event Calculus. 14th International Conference on Cooperative Information Systems - On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE, Nov 2006, Montpellier France, France. pp.408-425, 10.1007/11914853_25 . inria-00114023

HAL Id: inria-00114023
https://inria.hal.science/inria-00114023

Submitted on 15 Nov 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Web Service Mining and Verification of Properties: An approach based on Event Calculus

Mohsen Rouached, Walid Gaaloul, Wil M. P. van der Aalst, Sami Bhiri, and Claude Godart

LORIA-INRIA-UMR 7503
BP 239, F-54506 Vandœuvre-les-Nancy Cedex,France
{rouached,gaaloul,bhiri,godart}@loria.fr
Department of Technology Management, Eindhoven University of Technology P.O Box 513, NL-5600 MB, Eindhoven, The Netherlands
w.m.p.v.d.aalst@tm.tue.nl

Abstract. Web services are becoming more and more complex, involving numerous interacting business objects within complex distributed processes. In order to fully explore Web service business opportunities, while ensuring a correct and reliable execution, analyzing and tracking Web services interactions will enable them to be well understood and controlled. The work described in this paper is a contribution to these issues for Web services based process applications.
This article describes a novel way of applying process mining techniques to Web services logs in order to enable "Web service intelligence". Our work attempts to apply Web service log-based analysis and process mining techniques in order to provide semantical knowledge about the context of and the reasons for discrepancies between process models and related instances.

1 Introduction

With the ever growing importance of the service-oriented paradigm in system architectures, more and more (business) processes will be executed using service-oriented systems. Indeed, Service Oriented Architectures (SOA) seems to be a key architecture to support BPM (Business Process Management). In particular, with SOA, an application can be now considered as a composition of services, Workflow Management Systems (WfMSs), and legacy applications. Thus, a business process becomes a set of composed services that are shared across business units, organizations, or outsourced to partners.

Currently many products that offer modeling, analysis, and simulation facilities for such business processes exist. However, one of the great advantages offered by the coupling of BPM and SOA is that designers can not only model, analyze, simulate, but they can also use the result directly for deployment, using WSBPEL for instance. Functions at the modeling layer can be linked to

required services at the architecture level, and engines/systems can now manage the overall business process. This is a great improvement, and it clearly shows that BPM over SOA can add value over traditional WfMSs for instance.

However, there are many challenges for trully realizing BPM over SOA. A first challenge deals with the ability to adapt and to offer self-management of the designed processes [33]. This is an important topic since these processes are quite complex and dynamic, and deviations from the expected behavior may be highly desirable. In order to fully explore business opportunities while ensuring a correct and reliable behavior, the transition from vast amounts of Internet servers and Web Services transactions data to actionable modelling intelligence would be one of the major challenges in Web Services research field. The second challenge is the need for being able to check the consistency of the business process. This can be done both statically, i.e. at design time, or dynamically, i.e. at runtime. For the dynamic part of the verification, the business process should be auditable. For that, we can use process mining techniques, because processes (and their associated services) leave many traces of their behavior in the underlying systems they used to be executed.

Obviously, the practical benefit of mining techniques depends on the quality of the available log data. Though process execution logs can be used to reveal errors and bottlenecks, they do not provide any semantical information about the reasons for the observed discrepancies. In respect to the optimization of the process models these logs therefore provide only limited information to process engineers. Consequently, the PMS (Process Management System) is unaware of the applied deviations and thus unable to log information about them. The lack of traceability of process instance changes significantly limits the benefits of process mining and data mining approaches especially in dynamic and unpredictable environment such as in web service composition.

In this paper, unlike most process mining approaches, the emphasis is not on discovery. Instead we focus on verification, i.e., given an event log we want to verify certain specified properties, to provide knowledge about the context of and the reasons for discrepancies between process models and related instances. We focus on reasoning about event logs to capture the semantics of complex Web service combinations and checking their consistency. Since services behaviors and composition specifications we are modelling are event driven, our approach uses the Event Calculus (EC) of Kowalski and Sergot [14], and an extension proposed by Mueller on Discrete EC [18], to declaratively model event based requirements specifications. Compared to other formalisms, the choice of EC is motivated by both practical and formal needs, and gives several advantages. First, in contrast to pure state-transition representations, the EC ontology includes an explicit time structure that is independent of any (sequence of) events to model event-based interactions where a number of input events may occur simultaneously, and where the system behavior may in some circumstances be non-deterministic. Second, the EC ontology is close enough to existing types of event-based requirements specifications to allow them to be mapped automatically into the logical representation. More specifically, the EC ontology is close

enough to the WSBPEL specification to allow it to be mapped automatically into the logical representation. This allows us to use the same logical foundation for verification at both design time and runtime.

The remainder of the paper is structured as follows. Section 2 describes a running example used to illustrate our ideas. The log based verification is introduced in Section 3. Then, the existing web logging techniques are discussed and the used web logger is explained. Section 5 shows how the behavioral properties can be verified using the EC formalism. The related work is discussed in section 6. Finally, Section 7 concludes the paper and outlines some future directions.

2 Running Example

Throughout this article, we will illustrate our ideas using a running example of Web services composition. We consider a car rental scenario that involves four services. A Car Broker Service (CBS) acts as a broker offering its customers the ability to rent cars provided by different car rental companies directly from car parks at different locations. CBS is implemented as a service composition process which interacts with Car Information Services (CIS), and Customer Management Service (CMS). CIS services are provided by different car rental companies and maintain databases of cars, check their availability and allocate cars to customers as requested by CBS. CMS maintains the database of the customers and authenticates customers as requested by CBS. Each Car Park (CP) also provides a Car Sensor Service (CSS) that senses cars as they are driven in or out of car parks and inform CBS accordingly. The end users can access CBS through a User Interaction Service (UIS). Typically, CBS receives car rental requests from UIS services, authorizes customers contacting CMS and checks for the availability of cars by contacting CIS services, and gets car movement information from CSS services. However, many complications may arise. For example, CBS can accept a car rental request and allocate a specific car to it if, due to the malfunctioning of a CSS service, the departure of the relevant car from a car park has not been reported and, as a consequence, the car is considered to be available by the UIS service. Through this example, we aim to demonstrate how Web services logs can be specified and formalized in a way that enable the checking of some behavioral properties with respect to the composition process.

For the running example, we assume that we can log events such as the ones shown in Figure 1. Variables l_i, v_i, and c_i represent respectively the park number, the car number, and the customer identifier.

3 Log-based Verification

3.1 Formal Specification of Composition Properties

Our framework assumes service composition processes expressed in WSBPEL [2] and uses the Event Calculus [14] to specify the properties to be monitored. In this paper, the monitorable properties may include behavioural properties of

activity id	service originator	timestamp
...	...	...
Enter(v1,l1)	CSS	2006-03-13 10:40:39
RelKey(v1,c1,l1)	UIS	2006-03-13 10:40:44
Available(v1,l1)	CIS	2006-03-13 10:40:48
RetKey(v1,l1)	UIS	2006-03-13 10:40:54
Enter(v2,l2)	CSS	2006-03-13 10:40:57
...	...	...
CarRequest(c1,l2)	UIS	2006-03-13 10:41:01
FindAvailable(l2,veh)	CIS	2006-03-13 10:41:02
CarHire(c1,l2,v2)	UIS	2006-03-13 10:41:03
RetKey(v2,l2)	UIS	2006-03-13 10:41:09
...	...	...

Fig. 1. A fragment of the event log

the composition process and/or assumptions that service providers can specify in terms of events extracted from this specification. The behavioural properties are specified in terms of: (i) events which signify the invocation of operations in different services or the composition process and responses generated at the completion of these executions, (ii) the effects that these events may have on state variables of the composition (e.g., assignment of values), and (iii) conditions about the values of state variables at different time instances. The events, effects and state variable conditions are restricted to those which can be observed during the execution of the composition process. Assumptions are additional constraints about the behaviour of individual services in the execution environment. These constraints are specified by system providers and must be expressed in terms of events, effects and state variable conditions which are used in the behavioural properties directly or indirectly.

The behavioural properties of individual web services are extracted automatically from their WSDL descriptions and the WSBPEL specification of their composition process. Following the extraction of such properties, assumptions are specified by system providers in terms of event and state condition literals that have been extracted from the WSBPEL specification and, therefore, their truth-value can be established during the execution of the composition process. The extraction of behavioural formulas from WSBPEL specifications was done in a previous work BPEL2EC developed in [28]. The BPEL2EC is a tool that takes as input the specification of a web service composition expressed in WSBPEL and produces as output a possible behavioural specification of this composition in Event Calculus. This specification can be amended by service providers, who can also use the atomic formulas of the extracted specification to additional assumptions about the composition requirements if appropriate.

Once both behavioral properties and additional assumptions are formalized, we move to annotate the execution log with semantical information to enable reasoning on recorded events for checking the consistency of the above properties

and gathering reasons about deviations that may arise. This means that given an event log and an EC property, we want to check whether the observed behavior matches the (un)expected/(un)desirable behavior.

3.2 Formulating Properties: The EC Language

Assuming that the information system left a "footprint" in some event log, it is interesting to check whether certain properties hold or not. Before being able to check such properties, a concrete language for formulating dynamic properties is needed [34]. Given the fact that we consider behavioral properties where ordering and timing are relevant and we adopt an event driven reasoning, the Event Calculus (EC) [14] seems to be a solid basis to start from.

EC is a logic-based formalism for representing and reasoning about dynamic systems. We adapts a simple classical logic form of the EC, whose ontology consists of (i) a set of time-points isomorphic to the non-negative integers, (ii) a set of time-varying properties called fluents, and (iii) a set of event types (or actions). The logic is correspondingly sorted, and includes the predicates $Happens$, $Initiates$, $Terminates$ and $HoldsAt$, as well as some auxiliary predicates defined in terms of these. $Happens(a, t)$ indicates that event (or action) a actually occurs at time-point t. $Initiates(a, f, t)$ (resp. $Terminates(a, f, t)$) means that if event a were to occur at t it would cause fluent f to be $true$ (resp. $false$) immediately afterwards. $HoldsAt(f, t)$ indicates that fluent f is true at t. The auxiliary predicate $Clipped(t1, f, t2)$ expresses whether a fluent f was terminated during a time interval $[t1, t2]$. Similarly, the auxiliary predicate $Declipped(t1, f, t2)$ expresses if a fluent f was initiated during a time interval $[t1, t2]$. For using the previous predicates, we distinguish 4 different types of events in the context of web services:

1. The **invocation** of an operation by the composition process in one of its partner services. These events are represented by terms of the form: $ic.Service.OperationName(parameters)$.
2. The **return** from the execution of an operation invoked by the composition process in a partner service. These events are represented by terms of the form: $ir.Service.OperationName(parameters)$.
3. The **reply** following the execution of an operation that was invoked by a partner service in the composition process. These events are represented by terms of the form: $re.Service.OperationName(parameters)$.
4. The **assignment** of a value to a variable. These events are represented by terms of the form $as.AssignmentName(assignmentId)$.

The above types are assumed to have instantaneous duration. We also use fluents to signify the binding of specific variables of the composition process to specific values. Thus and by using the EC ontology, some behavioral properties of the services involved in the running example are specified as shown in Figure 2. This figure shows five properties expressed in EC. These are described below.

Property	EC specification
P1	$(\forall t1,t2)Happens(rc.CSS.Enter(oID1),t1)\wedge Initiates(rc.CSS.Enter(oID1),equalTo(v1,vID),t1)\wedge Initiates(rc.CSS.Enter(oID1),equalTo(p1,pID1),t1)\wedge Happens(rc.CSS.Enter(oID2),t2)\wedge(t1+t_m\leq t2)\wedge Initiates(rc.CSS.Enter(oID2),equalTo(v2,vID),t2)\wedge Initiates(rc.CSS.Enter(oID2),equalTo(p2,pID2),t2)\Longrightarrow(\exists t3)Happens(rc.CSS.Depart(oID3),t3)\wedge(t1+t_m\leq t3\leq t2-t_m)\wedge Initiates(rc.CSS.Depart(oID3),equalTo(v3,vID),t3)\wedge Initiates(rc.CSS.Depart(oID3),equalTo(p3,pID1),t3)$
P2	$(\forall t1,t2)Happens(ic.CIS.FindAvailable(oID,pID),t1)\wedge Happens(ic.CIS.FindAvailable(oID),t2)\wedge(t1\leq t2)\wedge HoldsAt(equalTo(availability(vID1),\text{"not avail"}),t2-t_m)\Longrightarrow\neg Initiates(ic.CIS.FindAvailable(oID),equalTo(vID2,vID1),t2)$
P3	$(\forall t1,t2,t3)Happens(ir.UIS.RelKey(oID1,vID),t1)\wedge Happens(ir.UIS.RelKey(oID1),t2)\wedge(t1\leq t2)\wedge Happens(ir.UIS.RetKey(oID2),t3)\wedge(t2\leq t3)\wedge Initiates(ir.UIS.RetKey(oID2),equalTo(v,vID),t3)\Longrightarrow(\forall t4)(t1<t4)\wedge(t4<t3)HoldsAt(equalTo(available(vID),\text{"not avail"}),t4)$
P4	$(\forall t1)Happens(ir.UIS.RelKey(v,c,l),t1)\wedge(\exists t2)(Happens(rc.CSS.Depart(v,l),t2)\wedge(t1\leq t2\leq t1+d*t_m))\Longrightarrow\neg(\exists t3)Happens(ic.CIS.Available(v,l)),t3)\wedge(t1+d*t_m\leq t3\leq t1+d*t_m))$
P5	$Happens(rc.UIS.CarRequest(c,l),t1)\wedge Happens(ic.CIS.FindAvailable(l,v3),t2)\wedge(t1\leq t2\leq t1+t_m)\wedge Initiates(ic.CIS.FindAvailable(l,v),equalTo(v,v3),t2)\Longrightarrow(\exists t3)Happens(ir.UIS.CarHire(c,l,v),t3)\wedge(t2\leq t3\leq t2+t_m)$

Fig. 2. Behavioural properties and assumptions of the running example

Property P1 is about the behavior of CSS services. The variable t_m refers to the minimum time between the occurrence of two events. Then, according to this property, if a car vID is sensed to enter a car park $pID1$ at time $t1$ and later at time $t2$ the same car is sensed to enter the same or a different car park, then a *Depart* event signifying the departure of vID from $pID1$ must have also occurred between the two *enter* events. The *Happens* predicates in P1 represent the invocation of the operations *Enter* and *Depart* in CBS by CSS following the entrance and departure of cars in car parks. The *Initiates* predicates in the same formula initiate fluents that represent the specific value bindings of the input parameters *vi* and *pi* (i=1,2,3) of the operations *Enter* and *Depart*.

Property P2, which is about the behavior of CIS services, indicates that the operation *FindAvailable*, which is provided by the CIS service and searches for available cars at specific car parks should not return the identifier of a car to CBS unless this car is available.

The property P3 states that whilst a customer has the key of a car, this car cannot be available for rental.

The property P4 specifies that when CBS receives the event $RelKey(v,c,l)$ that signifies the release of a car key to a customer, it waits for an event signifying the exit of the car from the car park for d time units (this message is to be sent by CSS). If the latter event occurs, CRS invokes the operation $Available(v,l)$ in CIS to mark the relevant car as unavailable.

4 Web Service Logging

4.1 Web service collecting solutions and Web mining log structure

In this section we examine and formalize the logging possibilities in service oriented architectures which is a requirement to enable the approach described in this paper. Thus, the first step in the Web Service mining process consists of gathering the relevant Web data, which will be analyzed to provide useful information about the Web Service behaviour. We discuss how these log records could be obtained by using existing tools or specifying additional solutions. Then, we show that the mining abilities is tightly related to what of information provided in web service log and depend strongly on its richness.

Existing logging solutions provide a set of tools to capture web services logs. These solutions remain quite "poor" to mine advanced web service behaviours. That is why **advanced logging solutions** should propose a set of developed techniques that allows us to record the needed information to mine more advanced behaviour. This additional information is needed in order to be able to distinguish between web services composition instances.

4.2 Existing logging solutions

There are two main sources of data for Web log collecting, corresponding to the interacting two software systems: data on the Web server side and data on the client side. The existing techniques are commonly achieved by enabling the respective Web servers logging facilities. There already exist many investigations and proposals on Web server log and associated analysis techniques. Actually, papers on Web Usage Mining WUM [25] describe the most weel-known means of web log collection. Basically, server logs are either stored in the *Common Log Format* [1] or the more recent *Combined Log Format* [2]. They consist primarily of various types of logs generated by the Web server. Most of the Web servers support as a default option the *Common Log Format*, which is a fairly basic form of Web server logging.

However, the emerging paradigm of Web services requires richer information in order to fully capture business interactions and customer electronic behavior in this new Web environment. Since the Web server log is derived from requests resulting from users accessing pages, it is not tailored to capture service composition or orchestration. That is why, we propose in the following a set of advanced logging techniques that allows to record the additional information to mine more advanced behavior.

4.3 Advanced logging solutions

Identifying web service composition instance : Successful mining for advanced architectures in Web Services models requires composition (choreog-

[1] http://httpd.apache.org/docs/logs.html
[2] http://www.w3.org/TR/WD-logfile.html

raphy/orchestration) information in the log record. Such information is not available in conventional Web server logs. Therefore, the advanced logging solutions must provide for both a choreography or orchestration identifier and a case identifier in each interaction that is logged.

A known method for debugging, is to insert logging statements into the source code of each service in order to call another service or component, responsible for logging. However, this solution has a main disadvantage: we do not have ownership over third parties code and we cannot guarantee they are willing to change it on someone else behalf. Furthermore, modifying existing applications may be time consuming and error prone.

Since all interactions between Web Services happen through the exchange of SOAP message (over HTTP), an other alternative is to use SOAP headers that provides additional information on the message's content concerning **choreography**. Basically, we modify SOAP headers to include and gather the additional needed information capturing **choreography** details. Those data are stored in the special <`WSHeaders`>. This tag encapsulates headers attributes like: `choreographyprotocol`, `choreographyname`, `choreographycase` and any other tag inserted by the service to record optional information; for example, the <`soapenv:choreographyprotocol`> tag, may be used to register that the service was called by $WS-CDL$ choreography protocol. The SOAP message header may look as shown in Figure 3. Then, we use SOAP intermediaries [3] which are an application located between a client and a service provider. These intermediaries are capable of both receiving and forwarding SOAP messages. They are located on web services provider and they intercept SOAP request messages from either a Web service sender or captures SOAP response messages from either a Web service provider. On Web service client-side, this remote agent can be implemented to intercept those messages and extract the needed information. The implementation of client-side data collection methods requires user cooperation, either in enabling the functionality of the remote agent, or to voluntarily use and process the modified SOAP headers but without changing the Web service implementation itself (the disadvantage of the previous solution).

Concerning **orchestration** log collecting, since the most web services orchestration are using a WSBPEL engine, which coordinates the various orchestration's web services, interprets and executes the grammar describing the control logic, we can extend this engine with a sniffer that captures orchestration information, i.e., the orchestration-ID and its instance-ID. This solution provides is centralized, but less constrained than the previous one which collects choreography information.

Using these advanced logging facilities, we aim at taking into account web services' neighbors in the mining process. The term neighbors refers to other Web services that the examined Web Service interacts with. The concerned levels deal with mining web service choreography interface (abstract process) through which it communicates with others web services to accomplish a choreography, or discovering the set of interactions exchanged within the context of a given choreography or composition.

$< soapenv : Header >$
$< soapenv : choreographyprotocol$
$soapenv : mustUnderstand = "0"$
$xsi : type = "xsd : string" > WS - CDL$
$< /soapenv : choreographyprotocol >$
$< soapenv : choreographyname$
$soapenv : mustUnderstand = "0"$
$xsi : type = "xsd : string" > OTA$
$< /soapenv : choreographyname >$
$< soapenv : choreographycase$
$soapenv : mustUnderstand = "0"$
$xsi : type = "xsd : int" > 123$
$< /soapenv : choreographycase >$
$< /soapenv : Header >$

Fig. 3. The SOAP message header

Collecting Web service composition Instance : The focus in this section is on collecting and analysing **single** web service composition instance. The issue of identifying several instances has been discussed in the previous section. The exact structure of the web logs or the event collector depends on the web service execution engine that is used. In our experiments, where we have used the engine bpws4j [3] uses log4j[4] to generate logging events. Log4j is an OpenSource logging API developed under the Jakarta Apache project. It provides a robust, reliable, fully configurable, easily extendible, and easy to implement framework for logging Java applications for debugging and monitoring purposes. The event collector (which is implemented as a remote log4j server) sets some log4j properties of the bpws4j engine to specify level of event reporting (INFO, DEBUG etc.), and the destination details of the logged events. At runtime bpws4j generates events according to the log4j properties set by the event collector. Below we show some example log4j 'logging event' generated by bpws4j engine.

```
2006-03-13 10:40:39,634  [Thread-35]  INFO   bpws.runtime - Outgoing res-
   ponse: [WSIFResponse:serviceID = '{http://tempuri.org/services/Custom-
   erReg}CustomerRegServicefb0b0-fbc5965758--8000'operationName = ''
   isFault = 'false'   outgoingMessage = 'org.apache.wsif.base.WSIFDefau-
   ltMessage@1df3d59 name:null  parts[0]:[JROMBoolean: : true]'
   faultMessage = 'null'  contextMessage = 'null']
2006-03-13 10:40:39,634  [Thread-35]  DEBUG  bpws.runtime.bus - Response
   for external invoke is[WSIFResponse:serviceID='{http://tempuri.org/se-
   rvices /CustomerReg}CustomerRegServicefb0b0-fbc5965758--8000'
     operationName = 'authenticate'  isFault = 'false'  outgoingMessage =
   org.apache.wsif.base.WSIFDefaultMessage@1df3d59 name:null  parts[0]:
   [JROMBoolean: : true]'faultMessage = 'null'  contextMessage = 'null']
2006-03-13 10:40:39,634  [Thread-35]  DEBUG  bpws.runtime.bus - Waiting
   for request
```

[3] http://alphaworks.ibm.com/tech/bpws4j
[4] http://logging.apache.org/log4j

The event extractor captures logging event and converts it to EC events by applying regular expressions. These expressions are described below.

Regular expressions for capturing events The following rules and assumptions are used in the regular expressions

R1 *$LogString* is the string representation of the 'logging event' received by the event receiver from the bpws4j engine
R2 *SubString($string, $substring)* extracts *$substring* at the beginning of *$string* where *$substring* can be a regular expression.
R3 *Matches($string, $substring)* returns true if *$substring* appears in *$string*, else returns false. *$substring* can be a regular expression.
R4 *StripEnds($string)* removes the first and the last character from *$string* and returns the *$string*.
R5 *$string1 + $string2* performs the concatenation of *$string1* and *$string2*.

Below, we precise how we use these rules for extracting EC formulas for two basic activities from WSBPEL:
Regular Expression for receive activity
$Happens(rc.ServiceName.operation(vID, var1, var2), t1)$ is an EC predicate that corresponds to a *receive* activity. A logging event from bpws4j that corresponds to a *receive* activity looks as follows,

```
2006-03-13 11:41:59,714[Thread-34]
   DEBUG  bpws.runtime.bus Invoking external service with[WSIFRequest:se
   rviceID='{http://tempuri.org/services/CarReg}CarRegServicefb0b0-fbc59
   65758-8000'operationName='isAvailable'incomingMessage='org.apache.wsi
   f.base.WSIFDefaultMessage@155423name:nullparts[0]:[JROMString:loc:One
   ]'contextMessage='null']
```

In this event [http8080-Processor25] is the unique ID assigned by the bpws4j engine to this instance of the *receive* activity and its corresponding *reply* activity. Applying the above assumptions and rules the event extractor generates the EC event as follows

```
if Matches($LoString, "http[0-9][0-9][0-9][0-9]-Processor[0-9]*") and
Matches($LogString, "operation") and Matches($LogString, "var1")
and Matches($LogString, "var2") then return
"Happens(rc.operation(" +  SubString($LogString, "http[0-9][0-9]
[0-9][0-9]-Processor[0-9]*") + StripEnds(SubString(SubString($LogString,
"var1: [0-9A-Za-z]*]"), " [0-9A-Za-z]*]")) + StripEnds(SubString(
SubString($LogString, "var2: [0-9A-Za-z]*]"), " [0-9A-Za-z]*])) + ")"
+ SubString($LogString, "[0-9][0-9][0-9][0-9]-([0][0-9]|[1][0-2])-[0-3]
[0-9] ([0-1][0-9]|[2][0-4]):[0-5][0-9]:[0-5][0-9],[0-9][0-9][0-9]) +
"R(SubString($LogString, "[0-9][0-9][0-9][0-9]-([0][0-9]|[1][0-2])-[0-3]
[0-9] ([0-1][0-9]|[2][0-4]):[0-5][0-9]:[0-5][0-9],[0-9][0-9][0-9]),
SubString($LogString,"[0-9][0-9][0-9][0-9]-([0][0-9]|[1][0-2])-[0-3]
[0-9]([0-1][0-9]|[2][0-4]):[0-5][0-9]:[0-5][0-9],[0-9][0-9][0-9]))" + ")"
```

The rest of the extraction schemes are analogous to those of *receive* activity. By doing this, the event log fragment 2 can be represented in formalised manner as shown in Figure 4.

L1 : $Happens(rc.CSS.Enter(v1, l1), 1)$
L2 : $Happens(rc.UIS.RelKey(v1, c1, l1), 5)$
L3 : $Happens(ic.CIS.Available(v1, l1), 9)$
L4 : $Happens(rc.UIS.RetKey(v1, l1), 15)$
L5 : $Happens(rc.CSS.Enter(v2, l2), 18)$
L6 : $Happens(rc.UIS.RetKey(v2, l2), 23)$
L7 : $Happens(ic.CIS.Available(v2, l2), 26)$
L8 : $Happens(rc.CSS.Enter(v1, l1), 27)$
L9 : $Happens(rc.UIS.RelKey(v2, c2, l2), 29)$
L10 : $Happens(ic.CIS.Available(v2, l2), 34$
L11 : $Happens(rc.UIS.CarRequest(c1, l2), 49)$
L12 : $Happens(ic.CIS.FindAvailable(l2, veh), 50)$
L13 : $Happens(ir.CIS.FindAvailable(l2, veh), 51)$
L14 : $Initiates(ir.CIS.FindAvailable(l2, v2), 51)$
L15 : $Happens(re.UIS.CarHire(c1, l2, v2), 52)$
L16 : $Happens(rc.UIS.RetKey(v2, l2), 54)$
...

Fig. 4. The CRS Event Log

5 Verifying Properties

5.1 The EC Checking

Given the properties specification shown in Figure 2 and the event log of Figure 4, the property P2 is found to be inconsistent with the expected behavior of CBS at t=54. The inconsistency arises because the literals L13 and L14 in Figure 2 and the literal $HoldsAt(equalTo(availability(v2), "not\ avail"), 50)$, which is derived from the literals L9 and L16 and the property P3 entail the negation of P2. In this example, the inconsistency is caused by the failure of the CSS service to send an $R.CSS.Depart(v2, l2)$ event to CBS following the event $Happens(Q.UIS.RelKey(v2, c2, l2), 28)$. Thus, according to P4, CBS invoked the operation *Available* to mark the vehicle v2 as available (see the literal L10 in Figure 4). Subsequently, when the operation $Q.CIS.FindAvailable(l2, v)$ was invoked in CIS (see literal L12), CIS reported v2 as an available vehicle. Note, however, that this inconsistency could only be spotted after the event signified by the literal L16 and by virtue of P4 (according to P4, a car whose key is released should not be considered as available until the return of its key).

One other case is that at t=54, the event L15 which was generated due to P5 can be detected as unjustified behavior. This is because this event can only

have been generated by P5. Note that, although in this case CBS has functioned according to P5, one of the conditions of this property is violated by the literal $Initiates(R.CIS.FindAvailable(l2, v), equalTo(v, v2), 51)$. This literal can be deduced from P2, the literal L13, and the literal $HoldsAt(equalTo(availability(v2), not\ avail), 50)$. The latter literal is deduced from L9 and L16 and assumption P3.

The property P2 is violated by the expected behaviour of CBS. According to this property, the operation *FindAvailable*, which is provided by the CIS service of CRS and searches for available cars at specific car parks should not return the identifier of a car to CBS unless this car is available. The violation of P2 in this case occurs since from P3 we can derive that $v2$ could not be available from T=30 when its key was released (see literal L9 in Figure 4) until T=53 (that is one time unit before its key was returned back - see literal L16 in Figure 4). Nevertheless, the execution of the operation *FindAvailable* of the CIS service at T=51 reported $v2$ as an available vehicle (see literal L14 in Figure 4).

5.2 Implementation Issues

In order to ensure an efficient satisfiability encoding for the EC, Mueller [19] presented an alternative classical logic axiomatization of the Event Calculus called *Discrete Event Calculus*, by restricting the timepoint sort to integers. Then, Mueller shows how Discrete Event Calculus problems can be encoded in first-order logic, and solved using a first-order logic automated theorem proving system, and developed a tool called the **Discrete Event Calculus Reasoner** (DEC Reasoner) [5].

Our approach has been implemented in Java and incorporates the following components: a requirements (behavioural properties and assumptions) editor, an event collector, a BPEL2EC tool, and a deviation viewer. Behavioural properties are extracted according to the patterns that we describe in [28] and represented in an XML-based language that we have defined to represent EC formulas.The properties extractor also identifies events, effects and state variables in the composition process that provide the primitive constructs for specifying further assumptions about the behaviour of the composition. These assumptions are specified by service providers using the assumption editor as shown in Figure 5.

The assumption editor offers to service providers the different types of events and fluent initiation predicates that have been identified in the composition process and supports the specification of assumptions as logical combinations of these event and fluent initiation predicates. Service providers may also use the editor to define additional fluents to represent services, service states, and relevant initiation and holding predicates. When an assumption is specified, the assumption editor can check its syntactic correctness. Figure 5 presents the steps in specifying assumptions using the assumption editor. The BPEL2EC tool is built as a parser that can automatically transform a given WSBPEL process into

[5] http://decreasoner.sourceforge.net

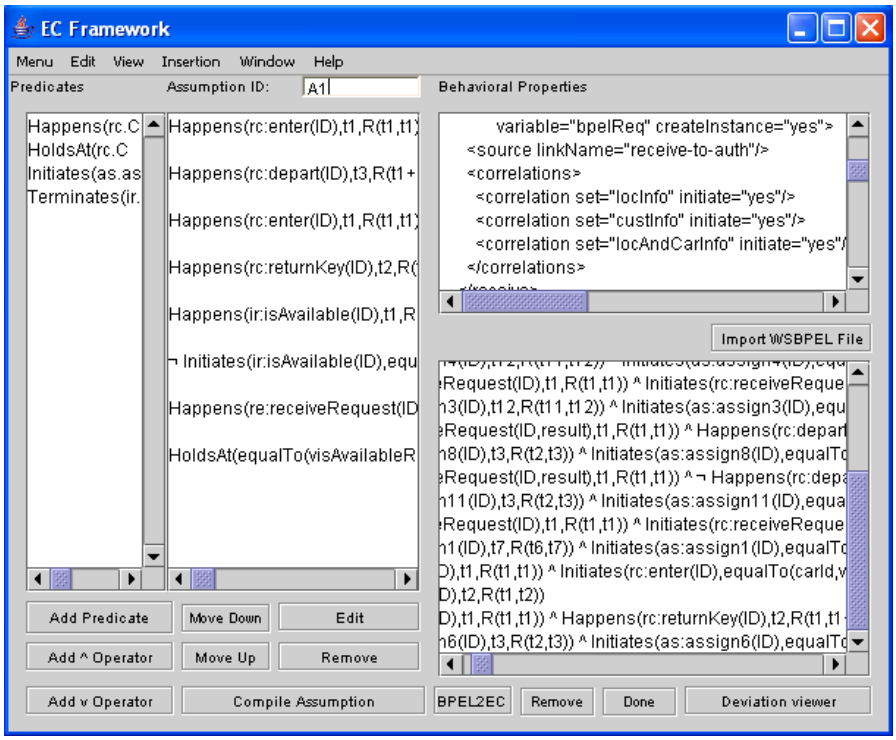


Fig. 5. The principal snapshot of the Monitoring Framework

EC formulas according to the transformation scheme detailed in [28]. It takes as input the specification of the Web service composition as a set of coordinated web services in WSBPEL and produces as output the behavioral specification of this composition in Event Calculus. This specification can be amended by the service providers to consider additional assumptions about the operations if appropriate.

While executing the composition process, the process execution engine generates events which are sent as string streams to the event collector of our framework. Irrelevant events are determined by the formulas that have been extracted or specified for monitoring by the service provider. Then, the EC checker (DEC for instance) processes the events which are recorded in the log by the event collector in the order of their occurrence, identifies other expected events that should have happened but have not been recorded (these events are derived from the assumptions by deduction), and checks if the recorded and expected events are compliant with the behavioural properties and assumptions of the composition process. In cases where the recorded and expected events are not consistent with these requirements, the EC checker records the deviation in a log deviations. The framework incorporates also a deviation viewer that is used to browse the detected violations of the formulas. A snapshot of this viewer is shown in Figure 6.

To evaluate our monitoring framework we performed a series of experiments in which we used an implementation of the CRS example as a case study. In this case study, we extracted 7 behavioural properties from the WSBPEL specifica-

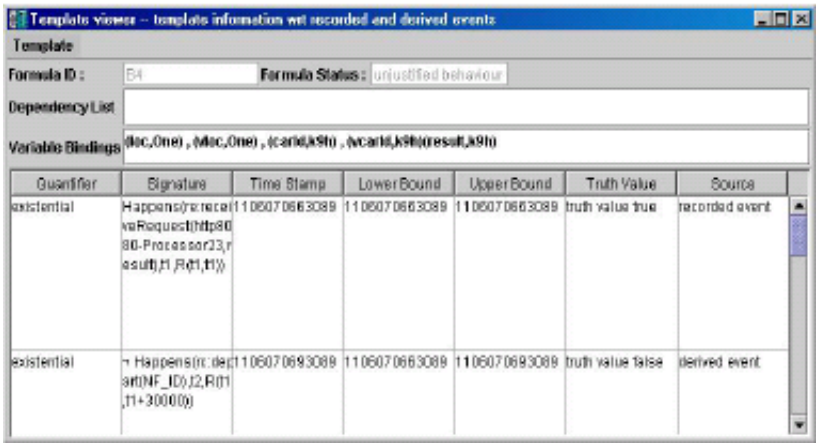


Fig. 6. Deviation Viewer

tion of the composition process of CRS and specified 4 assumptions. The WS-BPEL process of our case study and the behavioural properties and assumptions specified for it can be found at `http://www.loria.fr/~rouached/crs.zip`.

6 Related Work

Several attempts have been made to capture the behavior of BPEL [1] in some formal way. Some advocate the use of finite state machines [9], others process algebras [8], and yet others abstract state machines [7] or Petri nets [23,16,32]. Another branch of work concerning the area of "adapting Golog for composition of semantic web services" is carried out by Sheila McIlraith and others [17]. They have shown that Golog might be a suitable candidate to solve the planning problems occurring when services are to be combined dynamically at run-time. Additionally they propose to "take a bottom-up approach to integrating Semantic Web technology into Web services".

The need for monitoring web services has been raised by other researchers. For example, several research groups have been experimenting with adding monitor facilities via SOAP monitors in Axis `http://ws.apache.org/axis/`. [15] introduces an assertion language for expressing business rules and a framework to plan and monitor the execution of these rules. [4] uses a monitoring approach based on BPEL. Monitors are defined as additional services and linked to the original service composition. In [11,6], Dustdar et al. discuss the concept of web services mining and envision various levels (web service operations, interactions, and workflows) and approaches. Our approach fits in their framework and shows that web services mining is indeed possible, especially when using the existing process mining techniques. Related to the work in this paper is the work on conformance checking using Petri nets and event logs. In [35] abstract BPEL is mapped onto Petri nets and the resulting Petri net is compared with the event logs based on SOAP messages. ProM's conformance checker [29] is used to do

this comparison and the approach has been tested using Oracle BPEL. To give a complete overview of process mining, we refer to a special issue of Computers in Industry on process mining [38] and a survey paper [37]. Process mining can be seen in the broader context of Business (Process) Intelligence (BPI) and Business Activity Monitoring (BAM). In [12,30] a BPI toolset on top of HP's Process Manager is described. The BPI toolset includes a so-called "BPI Process Mining Engine". In [20] Zur Muehlen describes the PISA tool which can be used to extract performance metrics from workflow logs. In [24] a tool named the Web Service Navigator is presented to visualize the execution of web services based on SOAP messages. The authors use message sequence diagrams and graph-based representations of the system topology.

Formal verification of Web Services is addressed in several papers. The SPIN model-checker is used for verification [21] by translating Web Services Flow Language (WSFL) descriptions into Promela. [13] uses a process algebra to derive a structural operational semantics of BPEL as a formal basis for verifying properties of the specification. In [10], BPEL processes are translated to Finite State Process (FSP) models and compiled into a Labeled Transition System (LTS) in inferring the correctness of the Web service compositions which are specified using message sequence charts. In [22], Web services are verified using a Petri Net model generated from a DAML-S description of a service.

One common pattern of the above attempts is that they adapt static verification techniques and therefore violations of requirements may not be detectable. This is because Web services that constitute a composition process may not be specified at a level of completeness that would allow the application of static verification, and some of these services may change dynamically at run-time causing unpredictable interactions with other services.

The Event Calculus has been theoretically studied. Denecker et al. [5] use the Event Calculus for specifying process protocols using domain propositions to denote the meanings of actions. In [31] the Event Calculus has been used in planning. Planning in the Event Calculus is an abductive reasoning process through resolution theorem prover. [39] develops an approach for formally representing and reasoning about business interactions in the Event Calculus. The approach was applied and evaluated in the context of protocols, which represent the interactions allowed among communicating agents. Our previous work [27] is close enough to the current work. It presents an event-based framework associated with a semantic definition of the commitments expressed in the Event Calculus, to model and monitor multi-party contracts. This framework permits to coordinate and regulate Web services in business collaborations. Our paper [26] advocates an event-based approach for Web services coordination. We focused on reasoning about events to capture the semantics of complex Web service combinations. Then we present a formal language to specify composite events for managing complex interactions amongst services, and detecting inconsistencies that may arise at run-time.

7 Conclusion and Future Directions

The paper presents a novel approach that uses event logs to enable the verification of behavioral properties in web service composition. The properties to be monitored are specified in event calculus. The functional requirements are initially extracted from the specification of the composition process that is expressed in WSBPEL. This ensures that they can be expressed in terms of events occurring during the interaction between the composition process and the constituent services that can be detected from the log of the execution. Then, we have introduced the idea of Web services mining that makes use of the findings in the fields of process mining and apply these techniques to the context of Web services and service-oriented architectures. The main focus has not been put on discovery but on verification. This means that given an event log and a formal property, we check whether the observed behavior matches the (un)expected/(un)desirable behavior.

Future work will aim at the implementation of the approach in the context of ProM framework. Specifically, we will try to implement a plugin that permits to extract and adapt web service logs (using log4j for instance) to ProM's log format, and to express behaviour properties and assumptions about web service composition in a manner that enable the use (or possibly an extension) of ProM's verification plugins. Moreover, we are also aiming at languages that are more declarative and based on temporal logic. An example is the DecSerFlow: a Declarative Service Flow Language based on LTL [36].

References

1. T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business Process Execution Language for Web Services, Version 1.1. Standards proposal by BEA Systems, International Business Machines Corporation, and Microsoft Corporation, 2003.
2. A. Arkin, S. Askary, B. Bloch, and F.Curbera. Web services business process execution language version 2.0. Technical report, OASIS, December 2004.
3. M. Baglioni, U. Ferrara, A. Romei, S. Ruggieri, and F. Turini. Use soap-based intermediaries to build chains of web service functionality, 2002.
4. L. Baresi, C. Ghezzi, and S. Guinea. Smart Monitors for Composed Services. In *ICSOC '04: Proceedings of the 2nd International Conference on Service Oriented Computing*, pages 193–202, New York, NY, USA, 2004. ACM Press.
5. M. Denecker, L. Missiaen, and M. Bruynooghe. Temporal reasoning with abductive event calculus. In *Proceedings of the 10th European Conference and Symposium on Logic Programming (ECAI)*, pages 384–388, 1992.
6. S. Dustdar, R. Gombotz, and K. Baina. Web Services Interaction Mining. Technical Report TUV-1841-2004-16, Information Systems Institute, Vienna University of Technology, Wien, Austria, 2004.
7. D. Fahland and W. Reisig. ASM-based semantics for BPEL: The negative control flow. In D. Beauquier and E. Börger and A. Slissenko, editor, *Proc. 12th International Workshop on Abstract State Machines*, pages 131–151, Paris, France, March 2005.

8. A. Ferrara. Web services: a process algebra approach. In *ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing*, pages 242–251, New York, NY, USA, 2004. ACM Press.
9. J. Fisteus, L. Fernández, and C. Kloos. Formal verification of BPEL4WS business collaborations. In K. Bauknecht, M. Bichler, and B. Proll, editors, *Proceedings of the 5th International Conference on Electronic Commerce and Web Technologies (EC-Web '04)*, volume 3182 of *Lecture Notes in Computer Science*, pages 79–94, Zaragoza, Spain, Aug. 2004. Springer-Verlag, Berlin.
10. H. Foster, S. Uchitel, J. Magee, and J. Kramer. Compatibility verification for web service choreography. In *ICWS '04: Proceedings of the IEEE International Conference on Web Services (ICWS'04)*, page 738, Washington, DC, USA, 2004. IEEE Computer Society.
11. R. Gombotz and S. Dustdar. On Web Services Mining. In M. Castellanos and T. Weijters, editors, *First International Workshop on Business Process Intelligence (BPI'05)*, pages 58–70, Nancy, France, September 2005.
12. D. Grigori, F. Casati, M. Castellanos, U. Dayal, M. Sayal, and M. Shan. Business Process Intelligence. *Computers in Industry*, 53(3):321–343, 2004.
13. M. Koshina and F. van Breugel. Verification of business processes for web services. Technical report, New York University, SFUCMPT-TR-2003-06.
14. R. Kowalski and M. J. Sergot. A logic-based calculus of events. *New generation Computing 4(1)*, pages 67–95, 1986.
15. A. Lazovik, M. Aiello, and M. Papazoglou. Associating Assertions with Business Processes and Monitoring their Execution. In *ICSOC '04: Proceedings of the 2nd International Conference on Service Oriented Computing*, pages 94–104, New York, NY, USA, 2004. ACM Press.
16. A. Martens. Analyzing Web Service Based Business Processes. In M. Cerioli, editor, *Proceedings of the 8th International Conference on Fundamental Approaches to Software Engineering (FASE 2005)*, volume 3442 of *Lecture Notes in Computer Science*, pages 19–33. Springer-Verlag, Berlin, 2005.
17. S. McIlraith and T. Son. Adapting golog for composition of semantic web services. In *Proc of the 8th International Conference on Principles of Knowledge Representation and Reasoning*, 2002.
18. E. T. Mueller. Event calculus reasoning through satisfiability. *J. Log. and Comput.*, 14(5):703–730, 2004.
19. E. T. Mueller. Event calculus reasoning through satisfiability. *J. Log. and Comput.*, 14(5):703–730, 2004.
20. M. Mühlen and M. Rosemann. Workflow-based Process Monitoring and Controlling - Technical and Organizational Issues. In R. Sprague, editor, *Proceedings of the 33rd Hawaii International Conference on System Science (HICSS-33)*, pages 1–10. IEEE Computer Society Press, Los Alamitos, California, 2000.
21. S. Nakajima. Verification of web service flows with model-checking techniques. In *CW*, pages 378–385, 2002.
22. S. Narayanan and S. A. McIlraith. Simulation, verification and automated composition of web services. In *WWW '02: Proceedings of the 11th international conference on World Wide Web*, pages 77–88, New York, NY, USA, 2002. ACM Press.
23. C. Ouyang, W. Aalst, S. Breutel, M. Dumas, , and H. Verbeek. Formal Semantics and Analysis of Control Flow in WS-BPEL. BPM Center Report BPM-05-15, BPMcenter.org, 2005.
24. W. Pauw, M. Lei, E. Pring, L. Villard, M. Arnold, and J. Morar. Web Services Navigator: Visualizing the Execution of Web Services. *IBM Systems Journal*, 44(4):821–845, 2005.

25. J. Punin, M. Krishnamoorthy, and M. Zaki. Web usage mining: Languages and algorithms. In *Studies in Classification, Data Analysis, and Knowledge Organization*. Springer-Verlag, 2001.
26. M. Rouached and C.Godart. An event based model for web services coordination. In *2nd International Conference on Web Information Systems and Technologies (WEBIST 2006)*, pages 384–388. Setubal, Portugal, 11-13 April 2006.
27. M. Rouached, O. Perrin, and C. Godart. A contract-based approach for monitoring collaborative web services using commitments in the event calculus. In *Sixth International Conference on Web Information Engineering System (WISE05)*, pages 426–434, 2005.
28. M. Rouached, O. Perrin, and C. Godart. Towards formal verification of web service composition. In *Forth International Conference on Business Process Management (BPM06)*, 2006.
29. A. Rozinat and W. M. P. van der Aalst. Conformance testing: Measuring the fit and appropriateness of event logs and process models. In *Business Process Management Workshops*, pages 163–176, 2005.
30. M. Sayal, F. Casati, U. Dayal, and M. Shan. Business Process Cockpit. In *Proceedings of 28th International Conference on Very Large Data Bases (VLDB'02)*, pages 880–883. Morgan Kaufmann, 2002.
31. M. Shanahan and M. Witkowski. Event calculus planning through satisfiability. *J. Log. and Comput.*, 14(5):731–745, 2004.
32. C. Stahl. Transformation von BPEL4WS in Petrinetze (In German). Master's thesis, Humboldt University, Berlin, Germany, 2004.
33. W. M. P. van der Aalst, H. T. de Beer, and B. F. van Dongen. Process mining and verification of properties: An approach based on temporal logic. In *OTM Conferences (1)*, pages 130–147, 2005.
34. W. M. P. van der Aalst, H. T. de Beer, and B. F. van Dongen. Process mining and verification of properties: An approach based on temporal logic. In *OTM Conferences (1)*, pages 130–147, 2005.
35. W. M. P. van der Aalst, M. Dumas, C. Ouyang, A. Rozinat, and H. Verbeek. Choreography Conformance Checking: An Approach based on BPEL and Petri Nets (extended version). BPM Center Report BPM-05-25, BPMcenter.org, 2005.
36. W. M. P. van der Aalst and M. Pesic. Specifying, Discovering, and Monitoring Service Flows: Making Web Services Process-Aware. BPM Center Report BPM-06-09, BPMcenter.org, 2006.
37. W. M. P. van der Aalst, B. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. Weijters. Workflow Mining: A Survey of Issues and Approaches. *Data and Knowledge Engineering*, 47(2):237–267, 2003.
38. W. M. P. van der Aalst and A. Weijters, editors. *Process Mining*, Special Issue of Computers in Industry, Volume 53, Number 3. Elsevier Science Publishers, Amsterdam, 2004.
39. P. Yolum and M. P. Singh. Reasoning about commitments in the event calculus: An approach for specifying and executing protocols. *Annals of Mathematics and Artificial Intelligence*, 42(1-3):227–253, 2004.