

DISC-SET: Handling temporal and security aspects in the Web services composition

Ehtesham Zahoor, Olivier Perrin and Claude Godart
Université de Lorraine, Nancy 2, LORIA
BP 239 54506 Vandoeuvre-lès-Nancy Cedex, France

{ehtesham.zahoor, olivier.perrin, claude.godart}@loria.fr

Abstract—In this paper we propose the DISC-SET framework to handle the representation, solution computation and verification of temporal and security requirements in the services composition. The proposed approach provides a flexible event calculus based composition design, that allows for modeling different temporal (response time, time-units and other) and security aspects (access control, confidentiality and others) for Web services with different synchronization modes. The use of a formal approach allows to reason about and verify the security and temporal requirements. Further, as the proposed approach is integrated and builds upon the DISC framework, it allows to learn from run-time security and temporal constraints violations to take recovery actions.

I. INTRODUCTION

Web services are in the mainstream of information technology and are paving way for inter and across organizational application integration. Individual services may need to be composed and as the Web services are autonomous, having local (temporal and security) constraints and as the composition process may have some global (temporal and security) constraints, the need to represent and compute an ordering satisfying the associated constraints is evident.

The motivation of our work stems from the process modeling, analysis and monitoring in a flexible way such as needed by a crisis situation and the temporal and security properties are one of the most important aspects in such a situation. As the information comes from different sources with different local temporal constraints, it is challenging to find a plan which respects the local constraints of different participating services and conforms to the process-level global temporal constraints, and to achieve that in a flexible way by not over constraining the composition process. Further, the need to incorporate the security requirements in the composition process, such as confidentiality, integrity, access control and others is critical due to the ad-hoc nature of the process, and any process ordering should not compromise on the security requirements. In addition, the temporal and security aspects can be combined to have security requirements for a specific time interval, such as separation of duties (SoD) requirements for a particular time frame. Then the design time verification of the process to identify not only any control-based conflicts such as dead-locks, but also to identify conflicting security and temporal requirements, and

the run-time monitoring of the process while in execution are even more important as in a crisis situation the services are more error prone to the response time delays, network failures and other unforeseen situations. Further due to the critical nature of these processes, ideally the composition process should be able to recover from and should provide recovery actions and alternatives to handle such situations, preserving the associated security and temporal constraints.

In this paper we build upon the DISC (Declarative Integrated Self-healing web services Composition) framework [1] for the services composition, to propose the DISC-SET (DISC-extended with SEcurity and Timed properties) framework to handle the representation, computation and verification of temporal and security requirements in the services composition. When compared to traditional approaches, the proposed approach is declarative and allows for incorporation of security and temporal aspects (and their combinations) in a (possibly) partially defined process using a simple and flexible way, without over-constraining the process. Then the proposed approach is integrated as it allows to handles the temporal and security requirements at all stages of process life cycle, using the same event-calculus based model for composition design, verification and monitoring. At the process design stage, it allows for an expressive composition design incorporating various temporal (such as data and control based) and security (access control, data retention and others) aspects. The formal approach for composition design in turn allows for verifying the design for any data and control based conflicts, that need to be resolved for the successful execution of the process. Further, the proposed approach allows to monitor the process during execution and to take recovery actions in case of violations identified during the process execution.

II. RELATED WORK

In the literature, there have been many approaches that aim to handle different aspects of the Web services composition. However, the traditional approaches (such as WS-BPEL, WS-CDL) are highly procedural and they over-constrain the process, making it rigid and difficult to handle dynamically changing situations [1]. In contrast, some declarative approaches have been proposed [1], [2], [3], that allow for defining process in a flexible way by specifying

the constraints that mark the boundary of any acceptable solution to the composition process. The growing need for incorporating the security and temporal aspects in the services composition has led to many approaches that aim to handle security and temporal aspects in the services composition at different levels of process life-cycle. In general, the proposed approaches are either based on the procedural approaches such as BPEL, lack expressiveness and ease to model complex temporal and security aspects (or most importantly, their combination) or mostly focus on only part of the problem (not integrated to handle requirements at composition design, verification and monitoring stages).

The proposed approaches for incorporating temporal aspects include [4], [5], [6], [7] however, these approaches do not address the need for a unified framework and only focus on part of the problem. The proposed approaches also include [8], in which authors introduced a formalism called WSTTS to capture timed behavior of Web services and then using this formalism for model-checking WS-BPEL processes. In the planning domain, in [9] authors proposed Conditional Temporal Problem (CTP) formalism that allows for the construction of conditional plans that satisfy complex temporal constraints. The approaches also include ISDL [10] which uses time attributes to represent properties. In order to verify the timed properties authors proposed converting the BPEL process specification to timed automata and using UPPAAL model checker [11].

Further, there have been many approaches that aim to handle the security aspects in the Web services composition [12], [13], [14], [15] however, as similar to the approaches for incorporating the temporal aspects, they focus on only part of the problem. The approaches that deal with the representation of the security aspects and aim to incorporate the security requirements into the business process definition include [12], [16], [17]. Further, there have been approaches that aim to incorporate security requirements in the executable composition [13], [14], [18] or their enforcement at execution time [19], [12]. In [15] authors proposed to use a formalism that allows for incorporating security aspects at different levels of abstraction and has important contributions in terms of identification of security requirements in the services composition, however the approach is procedural as it is based on and extends BPMN notations, lacks formal representation and does not allow for verification of and reasoning about the security properties. The proposed **DISC-SET** framework serves as a unified integrated declarative framework for temporal and security properties representation, analysis, computation, allowing for solution re-computation to handle monitored violations.

III. MOTIVATING EXAMPLE

Let us consider a modified form of the emergency patient handling scenario [20]. After a serious road accident, the patient is in critical condition and no documents other than

his vehicle number are available to identify the patient. Patient is taken to nearby hospital in a remote region with limited resources and a composition process has been setup at the hospital to handle the emergency patient (see figure-1).

At the hospital, before the patient can be operated some blood tests are needed to identify any possible diseases, such as diabetes. Blood samples are thus sent to the laboratory department and the results are provided (pushed) to the requester as soon as they are available, normally between 4 to 10 minutes, by the push-based asynchronous Web service. Patient medical history can also be obtained by requesting the social security Web service, but for that the patient needs to be identified first. This can be done by contacting the vehicle and/or police department Web services to identify the owner information for the vehicle and to identify if the patient is indeed owner of the vehicle. These Web services support pull-based asynchronous invocation and thus the results can be pulled 5 minutes after sending the request. The patient information is then communicated to social security Web service to get the medical history for the patient. Further, additional blood supply can be requested by contacting the BloodBank Web service (see figure-1). Once patient medical history or lab results are known, the scheduling service should be contacted for both scheduling the operation theatre and the surgery team. As the hospital has limited facilities and as there is another surgery already planned after 90 minutes, the composition process introduces some global temporal constraints, including:

- The surgery should start within next 15 minutes and once the surgery has been scheduled, it needs to be confirmed 5 minutes before the surgery begins.
- The schedule request can only be made if patient history/ lab results and additional blood supply information is available.
- If there is some delay in obtaining the patient history/lab results, the delay can be notified to the scheduling service and the next surgery can either be rescheduled or transferred to some other surgery facility.

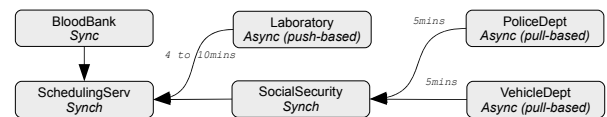


Figure 1. Motivating example

Further, due to the critical nature of the process, some security constraints need to be specified, including the following:

- Request messages to the services should be encrypted.
- The composition process is used by different users in two different roles, staff and admin. Staff users can only send requests and are not allowed to pull the response (and eventually read the confidential information), on the other hand admins can send the requests and receive response as well.

- All response messages are checked for integrity and are only valid (and deleted) 60 minutes after reception.

The emergency handling scenario presented above poses many challenges. First, specifying the exact sequence of activities as required by traditional procedural approaches, seems difficult as the solution depends on the specified constraints that can be difficult to solve manually. Then, the specification of the temporal and security (and other non-functional) aspects in a simple way without over-constraining the process is another challenge as the traditional procedural approaches are focused on the functional aspects. Further, if there is no solution then either the constraints are too strict or there is some conflict in the specified model that needs to be identified. Then, the process needs to be monitored during execution and in case of any violations to the security or temporal constraints, recovery actions should be taken to repair the process.

IV. THE PROPOSED **DISC-SET** FRAMEWORK

The proposed **DISC-SET** framework builds upon the **DISC** framework [1], which has three main stages, *Composition design*, *Instantiation and execution* and the *Composition monitoring*. The composition process starts when the user provides the *composition design* backed up by an event calculus model, by specifying the basic entities and associated constraints. The event calculus model for the composition design specified by the user can then be used to *instantiate and verify* the composition process. Then, the *composition monitoring* process detects, calculates side-effects and takes recovery actions to cater for and recover from the run-time violations.

The proposed **DISC-SET** framework extends the **DISC** framework [1] to add the security and timed properties representation, computation, verification, and re-computation (as a form of recovery action). In reference to **DISC** framework different aspects are handled at different stages, for the *representation* of timed and security properties we will extend the *composition design* phase to add the event calculus models for handling timed properties. Then, the *computation and verification* of timed and security properties can be handled in the *instantiation and execution* phase and finally the *re-computation* can be added as a recovery action in case of run-time violations captured in the *monitoring* phase of the **DISC** framework. In the sections to follow, we will discuss in detail different components of the **DISC-SET** framework.

V. REPRESENTATION

The proposed approach for the representation of timed and security properties is based upon event-calculus. Event calculus is a logic programming formalism for representing events and their side-effects (see [21], [1], [22] for a detailed discussion about event-calculus background and motivation

for its usage). The event calculus models are presented here using the discrete event calculus language [22] and we will only present the simplified models that represent the core aspects, intentionally leaving out the supporting axioms. All the variables (such as service, time) are universally quantified and in case of existential quantification, it is represented with variable name within curly brackets, {variablename}. Further, some event and fluent names are abbreviated in some models due to spacing issues.

A. Ground model

In this section we will first detail the ground model from the **DISC** framework [1], and will add temporal and security properties modeling in the sections to follow. At a basic level, the composition process can be regarded as the invocation/reception of response from the participating Web services. The basic event calculus model to handle services invocation is as below:

Ground model - CM-1.0

```
sort service   fluent ResponseReceived(service)   event InvokeService(service)
Initiates(InvokeService(service),ResponseReceived(service),time).
```

The basic entities in the model are Web services, they can be regarded as a sort in the discrete event calculus language terminology. Then we define an event to specify the service invocation *InvokeService(service)*, a fluent *ResponseReceived(service)*, which specifies if we have received the response message from the Web service and an axiom which states that if the action *InvokeService(service)*, happens at some time then the fluent *ResponseReceived(service)* continues to hold after that time. Before going further, let us discuss how this basic model can be used for reasoning purposes by using the model below:

```
sort service   service S1, S2   event InvokeService(service)
Initiates(InvokeService(service),ResponseReceived(service),time).

!HoldsAt(RespReceived(service), 0). ;initial situation
HoldsAt(ResponseReceived(service), 1). ;composition goal
```

In the model above, we add two instances of type service, called *S1* and *S2*, add initial condition that the fluent *ResponseReceived(service)* does not hold at time-point 0, a goal to the ground model above that the fluent must hold at time point 1 for services, and then invoke the reasoner. It gives us a plan, i.e. a temporal ordering, which shows that invoking the services concurrently at time-point 0, will result in receiving the response at time-point 1.

```
0 Happens(InvokeService(S1), 0). Happens(InvokeService(S2), 0).
1 +ResponseReceived(S1). +ResponseReceived(S2).
```

The ground model presented above does not model the response, request data from the services, and we can introduce the sorts named *request* and *response* to model it. We can further add the predicates, such as *RequestSource(request,*

service) and others and update the service invocation axiom in the ground model to cater for the newly added predicates. Using the modified model we can represent more complex orchestrations in which a service should be invoked multiple times with different request and response parameters, space limitations restrict us to discuss the model in detail. For simplicity, we will only consider the request, response parameters if they are needed, in the models to follow. Further, the invocation mode for the services can also be asynchronous and the process can either request and later "pull" the data from provider or alternatively data is "pushed" to the process by providers, when it is available.

B. Pull-based Asynchronous invocation

In order to model the pull-based asynchronous invocation, we update the model *CM-1.0*, and break down the invocation process by adding events and fluents for the sending request and then pulling the response. In the model below, we thus first introduce predicates that specify the synchronization mode for the Web service. Then we add another event to invoke asynchronous services and a new set of axioms to handle service invocation and then pulling for the response.

Asynchronous invocation (pull-based) - extends *CM-1.0*

fluent *ResponseRequested(service)*
 event *ReceiveResponsePull(service), InvokeAsynchService(service)*
 predicate *IsSynchronous(service), IsASynchronousPull(service)*

Initiates(InvokeAsynchService(service), ResponseRequested (service),time).
Initiates(ReceiveResponsePull(service), ResponseReceived (service),time).
Happens(ReceiveResponsePull(service), time1) → {time2}HoldsAt (ResponseRequested(service), time2) & time1 > time2.

Happens(InvokeAsynchService(service),time1) & Happens(ReceiveResponsePull (service), time2)→ time2 - time1 >= 20.

The second last axiom specified in the above model specifies that the event *ReceiveResponsePull(service)* can only happen if we have already requested for the response. Other axioms in the model specify the response request and the eventual "pull" for the response message. The last axiom models the minimum time after which the response data is available to be pulled and this information can be imported from the service repositories.

C. Push-based Asynchronous invocation

In order to model the push-based asynchronous invocation, we introduce the queues that can be used to store the pushed data from the service providers and composition process can then use the data from these queues. We use the pull-based asynchronous model and add the fluent and corresponding event to model the queues and data being pushed to queues by the service providers. In the updated model, the process first sends the request, *InvokeAsynchService(service)*, and then the response is pushed to the process queue, *PushResponse(service)*, between the specified time intervals. Once the data is

available in the queues, *HoldsAt(ResponsePushed(service), time2)*, the response can then be retrieved from the process queue, *ReceiveResponsePush(service)*.

Asynchronous invocation (push-based) - extends *CM-1.0*

fluent *ResponsePushed(service)* event *PushResponse(service), ReceiveResponse-Push(service)*
Initiates(PushResponse(service),ResponsePushed(service),time).
Initiates(ReceiveResponsePush(service),RespReceived (service),time).

Happens(ReceiveResponsePush(service), time1) → {time2} Hold-At(ResponsePushed(service), time2) & time1 > time2.

D. Modeling temporal aspects

The temporal aspects in the services composition can be broadly categorized into data and control based temporal properties. Below we briefly discuss event calculus modeling for some of the temporal aspects.

Response time: For modeling the response time of asynchronous services, it is sufficient to introduce the delay between the service invocation and later pull/push for the response (first axiom in the model below). However, in case of synchronous services, we can model it by breaking down the invocation process in two events, *StartInvoke* and *EndInvoke*, as modeled in the second axiom below.

Happens(InvokeASynchService(S1),time1) & Happens (ReceiveResponsePull(S1),time2) → time2 -time1 = 10.
Happens(StartInvoke(S1), time1) & Happens(EndInvoke(S1), time2) → time2 - time1 = 10.

Refresh: The temporal properties include the refresh constraint which requires the service re-invocation after a fixed time, for instance to invoke some service (*S1*) every 2 minutes. The refresh operation when applied to data flow, requires the data to be re-fetched once it expires. The first event calculus axiom below models the control-flow based refresh constraint while the second axiom models the data flow version, assuming the event *InvalidateData* results in data expiry.

Happens(InvokeSynchService(S1),time1)&time2-time1=2→Happens (InvokeSynchService(S1),time2).
Happens(InvalidateData(S1), time) → Happens(InvokeSynchService(S1),time).

Invocation time-frame: The invocation timeframe constraint requires a service to be invoked within a fixed timeframe, for instance to invoke a service *S1* between 10 and 20 minutes after some event happens). The timeframe constraint when applied to data flow requires that the data from a service is available (at-least/only) between the specified time-frame. The first axiom below models the control-flow based invocation time-frame constraint while the second axiom models the data flow version.

Happens(SomeEvent(), time1) → {time2} time2 > time1 + 10 & time2 < time1 + 20 & Happens(InvokeSynchService(S1), time2).
Happens(SomeEvent(), time1) & time2 > time1 + 10 & time2 < time1 + 20 → HoldsAt(ResponseReceived(S1), time2).

Happens(InvokeSynchService(S1), 10). HoldsAt(ResponseReceived(S1), 10).

Further, a variant of execution time-frame aspect is to invoke a service at exact time point (such as exactly at 10 minutes). When applied to data, it requires the data to be available at (at-least/only) specified time-point. The last axiom in the above model, handles this behavior.

Invocation delay: The execution delay aspect requires that the successive invocations of the service must be delayed by some time (possibly to prevent overloading a service). This constraint can also be specified to specify the invocation delay between multiple services, such as *S2* should be invoked exactly three minutes after the service *S1*. The first axiom below models the invocation delay for invocations of a single service, while the second adds the delay between the invocation of multiple services.

$$\text{Happens}(\text{InvokeSynchService}(S1), \text{time1}) \ \& \ \text{Happens}(\text{InvokeSynchService}(S1), \text{time2}) \ \& \ \text{time1} \neq \text{time2} \ \rightarrow \ \text{time2} - \text{time1} = 2.$$

$$\text{Happens}(\text{InvokeSynchService}(S1), \text{time1}) \ \& \ \text{Happens}(\text{InvokeSynchService}(S2), \text{time2}) \ \& \ \text{time1} \neq \text{time2} \ \rightarrow \ \text{time2} - \text{time1} = 3.$$

Modeling time-units: In order to model the Web services with temporal constraints in different time-units, we need to add semantics to the event calculus time-points, so that saying that an event happens at time-point 1 can signify one second/minute and so on. The possible solutions to this problem include to convert all the time-units to some common time-unit (such as seconds), however a smaller common time-unit such as seconds, and converting all other units to seconds (10 minutes equals 600 seconds) will increase the resulting event-calculus to SAT encoding size and thus is not feasible. On the other hand, converting all the units to a higher common format such as minutes will not allow to handle the smaller time units, such as seconds, as the *DECReasoner* is discrete. As an alternative, we pre-process the time-units associated with different participating services to find a solution and then post-process the solution returned to update the time-units associated with each service. Space limitations restrict us to detail the process further.

E. Modeling security aspects

In terms of modeling security aspects using event-calculus, we have proposed the dynamic authorization policies for task delegation [23] and in this work, we use a set of security requirements identified in [15] and discuss how the proposed framework can be used to model and reason about the security requirements. The security requirements include confidentiality, data retention, access control, authentication, data integrity and others (see [15] for a detailed discussion). They can be represented by defining the event calculus fluents and events that have impact on the fluents. Below we discuss some of security aspects and their modeling in the event calculus:

Confidentiality security property requires that the critical information, such as credit card and other personal information, should be encrypted and protected from unauthorized access. In event calculus, the *confidentiality* property can be modeled as a fluent, *IsConfidential(response)*, which is set by the event *Encrypt(response)*, and defines that as the *Encrypt* event happens the fluent continues to hold and the data is considered confidential. In the model below, the *Initiates* axiom handles this behavior:

$$\text{fluent } \text{IsConfidential}(\text{response}) \quad \text{event } \text{Encrypt}(\text{response}) \\ \text{Initiates } (\text{Encrypt}(\text{response}), \text{IsConfidential}(\text{response}), \text{time}).$$

Data retention property associates a time-to-live (TTL) information with the data requiring it to be deleted after a certain time and it can be modeled as similar to data validity model discussed during the streaming Web services modeling [1]. Further, the **data integrity** security property requires that the sensitive data (such as personal information of a user) has to be verified for data corruption before usage. The *data integrity* can be modeled as a fluent, *IsValid(response)*, which is activated by the event *CheckDataIntegrity(response)*. Then the **authentication** property requires that only appropriate users have access to the sensitive or critical information held by the services. The event calculus model below handles the authentication and data integrity security requirement, however one important difference is that the fluent is released (*Releases*) instead of initiated (*Initiates*) and this highlights that the effect of the event to check the data integrity (and authentication) is either that the data integrity holds (user is authenticated) or is considered corrupted (authentication fails). The *Releases* axioms model this behavior:

$$\text{fluent } \text{IsValid}(\text{response}), \text{IsAuthenticated}(\text{user}) \\ \text{event } \text{CheckDataIntegrity}(\text{response}), \text{Authenticate}(\text{user}) \\ \text{Releases } (\text{CheckDataIntegrity}(\text{response}), \text{IsValid}(\text{response}), \text{time}). \\ \text{Releases } (\text{Authenticate}(\text{user}), \text{IsAuthenticated}(\text{user}), \text{time}).$$

Auditing property requires that all the operations performed by the composition process should be logged and are available for auditing (if needed). In relation to the proposed event calculus modeling approach, an event called *CreateLog()* can be used that should be invoked after each event to log the process state. Then the **access control** restricts the access to resources to only the authorized users and in the literature different access control schemes have been proposed (such as *RBAC*, *TBAC* and others). Space limitations restrict us to discuss the basic role based access control (RBAC) in this work. For the proposed modeling approach, different roles (organized in hierarchy) can be modeled by creating EC sort/sub-sort for each role. Tasks can then be assigned/delegated and access is controlled using EC predicates, fluents and axioms. We can create a sort named *user* that models the users of the system and a sort named *role* which represents the role(s) to which a user

belong by using the predicate $HasRole(user, role)$. Then for each event requiring controlled access, we can define an predicate called $HasRestrictedAccess(event, role)$, which serves as the role based access control assignment for the event. Finally we can specify an axiom that limits access to only those user that belong to the particular role. We will discuss an example of access control in the section-VI.

```
sort user, role
predicate HasRole(user, role), HasRestrictedAccess(event, role)
Happens(SomeEvent(..., user), time) & HasRestrictedAccess(SomeEvent(..., user),
role) & HasRole(user, role1) → role = role1.
```

F. Example

Let us now review the motivating example and discuss the event calculus model with temporal and security properties representation. In order to keep the model simple we consider the multiple invocations of $SchedulingService$ with different parameters, as multiple service invocations. In the model below we first define the instances of the sort $service$ that specify the Web service instances, their synchronization modes and local temporal constraints:

```
service SocialSecurity, VehicleDept, ReSchedulingServ ...
IsSynchronous(SocialSecurity), IsPullBasedASynchronous (VehicleDept), ...
Happens(InvokeAsynchService(VehicleDept), time1) & Happens (ReceiveResponsePull(VehicleDept), time2) → time2 - time1 = 10...
```

Next, we introduce the dependencies between different services, $SocialSecurity$ service has dependency on either $Police$ or $VehicleDept$ service, while the $SchedulingService$ has dependency on either $SocialSecurity$ or $Laboratory$ services. We can model the dependencies between $SchedulingService$ and scheduling confirmation service in a similar fashion.

```
Happens(InvokeSynchService(SocialSecurity),time1) → {time2} (HoldsAt(ResponseReceived (VehicleDept),time2) | HoldsAt (ResponseReceived (PoliceDept),time2)) & time1 >= time2.
Happens(InvokeSynchService(SchedulingServ),time1) → {time2} (HoldsAt(ResponseReceived (SocialSecurity),time2) | HoldsAt (ResponseReceived (Laboratory), time2)) & time1 >= time2. ...
```

```
fluent IsConfidential(response), event Encrypt(response)
Initiates(Encrypt(response),IsConfidential(response),time).
Happens(InvokeService(service,request),time1) → {time2} HoldsAt(IsConfidential (request,time2)&time1 >= time2. ...
```

The model above also models the security aspects and specifies that all the request messages should be confidential i.e. encrypted, before the request can be sent to the Web services. We have omitted the data integrity models due to the space limitations. Next, we model the access control using the proposed framework by creating two users $SomeUserA$, $SomeUserB$ and assigning them different roles and then restring the access for events:

```
user SomeUserA, SomeUserB role Staff, Admin
HasRole(SomeUserA, Staff) HasRole(SomeUserB, Admin)
Happens(InvokeSynchService(service,request,user),time) & HasRestrictedAccess (InvokeSynchService(service, request, user), role) & HasRole(user, role1) → role = role1.
```

```
!HoldsAt(ResponseRequested(service),0). !HoldsAt(RespReceived(service),0)...
HoldsAt(RespReceived(SchedulingConServ),10) | HoldsAt( RespReceived(ReSchedulingServ),10).
```

In the model above, last two axioms specify the initial situation for the fluents, that they do not hold at time point 0, and the goal for the process. As the surgery must start in 15 minutes, so we need to either confirm/reschedule other surgery at time-point 10, we thus specify a goal for the composition process that the response from either of these services is available at time-point 10.

VI. COMPUTATION AND VERIFICATION

The specified composition design with associated temporal and security constraints can then be used for the solution computation or for the process verification. In reference to the proposed implementation architecture [1], the event calculus model is encoded into a SAT problem and the SAT solver is invoked to provide a set of solutions to the SAT problem. However, if there are some conflicts in the composition design and/or the specified constraints are too strict, this leads to empty solution set and requires the verification of the composition design to identify any conflicts or hard constraints.

With respect to the process verification, the proposed framework allows for both design-level verification and for the run-time monitoring (and recovery) using the different reasoning techniques on the event-calculus based composition design. *Abduction reasoning* is used to find a set of solutions and to identify any conflicts, while *deduction reasoning* is used to calculate the effect of run-time violations (more details can be found in [1]). The proposed approach to the design-time verification relies on the SAT solver to provide a set of near-miss models and/or unsatisfied clauses and as underlined in our previous work ([1]), delegation of verification task to the SAT solver has many benefits. First, the reasoner we are using transforms the EC model into a SAT problem, thus the same SAT encoding can be also used for both solution finding and verification purposes. Then, it allows not only for the conflicts (such as deadlocks) detection, but allows for identifying the hard constraints that should be relaxed to find a solution and for identifying other side-effects such as the data expiry and others. Then, it provides an highly extensible approach, same SAT encoding can be either analyzed by multiple solvers. In reference to the proposed approach, the properties that can be verified include the temporal aspects such as minimum and maximum time intervals that exist between the execution of one or several services, data retention such as the maximum time interval about data validity and security aspects such as the SoD constraint requiring prohibition to invoke of a service if another service had been executed, possibly combined with temporal conditions (e.g. the ban lasts only two hours), access control aspects such as the permission/prohibition to invoke a service given a role. These requirements are termed as invariants and are handled by adding axioms to the

event calculus based composition design. Below we briefly discuss some of the security and temporal properties (and their combinations) verification requirements that can be handled using the proposed approach, we will also briefly discuss the event-calculus axioms that should be added to the composition design for these requirements.

- *Is there any solution exists that satisfies the associated temporal and security (SoD, access control, ...) constraints?* This can be verified by invoking the reasoner. In case of empty solution set, a list of unsatisfied clauses or partial plan is returned by the SAT solver.
- *In case of response time delay, re-plan to find alternatives to the current execution plan.* This requirement requires the process to recover from a run-time monitored violation and find some alternatives to the current plan. As the proposed approach is integrated, it allows to recover from run-time violations (see [1]) and the monitored violation can be based on security such as data not remains confidential, access to unauthorized data and corresponding actions can include to terminate the process or send alert for the violation.

$Happens(Invoke(service),time1) \ \& \ !HoldsAt(RespRecvd(service), time2) \ \& \ time2 - time1 != 10 \ \rightarrow \ Happens(Replan(),time2).$

- *If data does not remain valid, re-invoke the (idempotent) service.* This requirement can be handled at both design/execution time and an event calculus axiom can be used to re invoke the service as soon as data is not valid.

$Happens(InvalidData(service),time) \ \rightarrow \ Happens(Invoke(service),time).$

- *Is there any conflict in the security policy specification?* This requirement can be handled by using event calculus axioms that can check if the users have simultaneously conflicting roles, or if the user belongs to one role it can-not belong to some other role, or in case of delegation of roles it cannot have some other role for some specific time-interval.

Let us now review the motivating example and discuss the solution computation. Invoking the reasoner for the event calculus model for the motivating example gives us a set of models including the following:

```

0  Happens(Encrypt(SomeRequest), 0).
1  +IsConfidential(SomeRequest).
   Happens(InvokeAsynchService(Laboratory,SomeRequest,SomeUserA), 1).
   ... Invocation for PoliceDept and VehicleDept
   Happens(InvokeSynchService(BloodBank, SomeRequest, SomeUserA), 1).
2  +ResponseReceived(BloodBank). +ResponseRequested(Laboratory). ...
   ...
6  Happens(ReceiveResponsePull(PoliceDept, SomeUserB), 6).
7  +ResponseReceived(PoliceDept).
   Happens(InvokeSynchService(SocialSecurity, ...,7).
8  +ResponseReceived(SocialSecurity).
   Happens(InvokeSynchService(SchedulingServ, ...,8).
9  +ResponseReceived(SchedulingServ).
   Happens(InvokeSynchService(SchedulingConServ,...),9).
10 +ResponseReceived(SchedulingConServ).
```

The model above shows that there exists a solution in which the response can be received from *police/vehicle department* Web service for identifying the patient and thus retrieving the patient history from *social security Web service* and in turn *scheduling* the patient. The *laboratory* Web service can take 4-10 minutes and in the worst case scenario the results are obtained after 10 minutes, so it is not chosen as a possible solution. However, the proposed solution is based on design level service contracts and they may change at the execution time for the actual service invocations. In reference to the motivating example, let us assume that the patient history is not available as the services are not respecting design time service agreements and thus at monitoring time a violation is detected at time-point 6. To handle this violation the action specified by the user may be to recompute the plan and reasoner is invoked again after adding the updated information to the plan. There are two cases, as the *laboratory* service takes 4-10 minutes the result can be obtained after 4 minutes. So if we consider this case the following regenerated plan can be used:

```

5  ...
   Happens(PushResponse(Laboratory), 5).
6  +ResponsePushed(Laboratory).
   ...
8  +ResponseReceived(Laboratory). ...
9  Happens(InvokeSynchService(SchedulingConServ,SomeRequest,SomeUserA),9).
10 +ResponseReceived(SchedulingConServ).
```

However, if the data is not yet available from the laboratory, the request to reschedule the next surgery should be sent and invoking the reasoner gives us the following plan:

```

9  ...
   Happens(InvokeSynchService(ReSchedulingServ,SomeRequest,SomeUserA),9).
10 +ResponseReceived(ReSchedulingServ).
```

VII. PERFORMANCE EVALUATION

The event-calculus models presented in this work use the discrete event calculus language [22] and they can directly be used for reasoning purposes. The performance evaluation tests were conducted on a *MacBook Pro* with Intel core 2 Duo 2.53 Ghz processor and 4GB RAM running *Mac OS-X 10.6*. The *DECReasoner* version 1.0 and the SAT solver, *relnat-2.0/zchaff* were used for reasoning. The performance results are highlighted in figure-2. In order to test the scalability of the approach, the number of services is obtained by replicating the motivating example (which has 8 services) with all the dependencies and constraints, multiple times. So 16 services is the motivating example replicated twice and so on. We evaluate three different test cases for *solution computation*, *verification* and *re-computation* and for each test case, the total time taken by the reasoner is divided into time for *encoding* event-calculus to SAT problem and solution finding by the SAT solver.

The evaluation results show that the encoding process does not scale very well, but it is still reasonable given the large size of the problem. Moreover, we are in the process of optimizing this encoding process. The solution computation by using the *relnat* solver is very efficient and

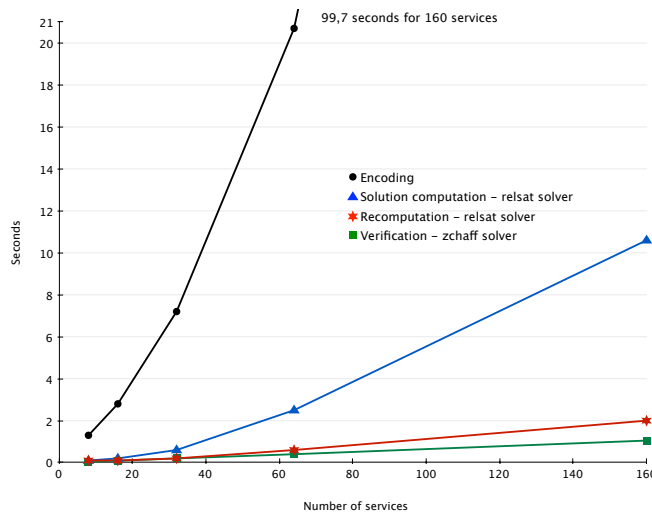


Figure 2. Performance evaluation

the zchaff solver used for the verification performs even better. Then for the solution re-computation to cater for run-time violations, the time taken to find the solution drops, and this highlights that the part of the solution is already known (in terms of partial plan added to the model) and thus it requires less work by the solver.

VIII. CONCLUSION

In this paper we build upon the **DISC** framework for the services composition, to propose the **DISC-SET** framework to handle the representation, computation and verification of temporal and security aspects in the services composition, in a declarative and integrated way. The proposed approach provides a flexible event calculus based composition design, that allows for modeling different aspects such as the temporal and security constraints for Web services with different synchronization modes. Further, the approach allows to reason about and perform both design-time verification and run-time monitoring of the composition process. We have identified a set of requirements for the process verification and discussed how the proposed model can handle them. We have also presented a motivating example and performance evaluation results, that highlight our approach.

REFERENCES

- [1] E. Zahoor, O. Perrin, and C. Godart, "Disc: A declarative framework for self-healing web services composition," in *ICWS*, 2010.
- [2] W. M. P. van der Aalst and M. Pesic, "Decserflow: Towards a truly declarative service flow language," in *The Role of Business Processes in Service Oriented Architectures*, 2006.
- [3] M. Pesic and W. M. P. van der Aalst, "A declarative approach for flexible business processes management," in *Business Process Management Workshops*, 2006.
- [4] N. Guermouche and C. Godart, "Asynchronous timed web service-aware choreography analysis," in *CAiSE*, 2009.

- [5] L. Bordeaux, G. Salaün, D. Berardi, and M. Mecella, "When are two web services compatible?" in *TES*, 2004, pp. 15–28.
- [6] J. Ponge, B. Benatallah, F. Casati, and F. Toumani, "Fine-grained compatibility and replaceability analysis of timed web service protocols," in *ER*, 2007.
- [7] B. Benatallah, F. Casati, J. Ponge, and F. Toumani, "On temporal abstractions of web service protocols," in *CAiSE Short Paper Proceedings*, 2005.
- [8] R. Kazhamiakin, P. K. Pandya, and M. Pistore, "Representation, verification, and computation of timed properties in web," in *ICWS*, 2006, pp. 497–504.
- [9] I. Tsamardinos, T. Vidal, and M. E. Pollack, "Ctp: A new constraint-based formalism for conditional, temporal planning," *Constraints*, vol. 8, no. 4, pp. 365–388, 2003.
- [10] D. A. C. Quartel, R. M. Dijkman, and M. van Sinderen, "Methodological support for service-oriented design with isdl," in *ICSOC*, 2004, pp. 1–10.
- [11] N. Guermouche and C. Godart, "Timed model checking based approach for web services analysis," in *ICWS*, 2009.
- [12] M. Menzel, I. Thomas, and C. Meinel, "Security requirements specification in service-oriented business process management," in *ARES*, 2009, pp. 41–48.
- [13] D. A. Basin, J. Doser, and T. Lodderstedt, "Model driven security: From uml models to access control infrastructures," *ACM Trans. Softw. Eng. Methodol.*, vol. 15, no. 1, 2006.
- [14] D. Z. G. Garcia and M. B. F. de Toledo, "Ontology-based security policies for supporting the management of web service business processes," in *ICSC*, 2008.
- [15] A. R. R. Souza, B. L. B. Silva, F. A. A. Lins, J. C. Damasceno, N. S. Rosa, P. R. M. Maciel, R. W. A. Medeiros, B. Stephenson, H. R. M. Nezhad, J. Li, and C. Northfleet, "Incorporating security requirements into service composition: From modelling to execution," in *ICSOC/ServiceWave*, 2009.
- [16] T. Neubauer and J. Heurix, "Defining secure business processes with respect to multiple objectives," in *ARES*, 2008.
- [17] A. Rodríguez, E. Fernández-Medina, and M. Piattini, "A bpmn extension for the modeling of security requirements in business processes," *IEICE Transactions*, vol. 90-D, 2007.
- [18] S. Chollet and P. Lalanda, "Security specification at process level," in *IEEE SCC (1)*, 2008, pp. 165–172.
- [19] H. Song, Y. Sun, Y. Yin, and S. Zheng, "Dynamic weaving of security aspects in service composition," in *SOSE*, 2006.
- [20] E. Zahoor, O. Perrin, and C. Godart, "Mashup model and verification using mashup processing network," in *CollaborateCom2008*. ACM, 2008.
- [21] R. A. Kowalski and M. J. Sergot, "A logic-based calculus of events," *New Generation Comput.*, vol. 4, no. 1, 1986.
- [22] E. T. Mueller, *Commonsense Reasoning*. Morgan Kaufmann Publishers Inc., 2006.
- [23] K. Gaaloul, E. Zahoor, F. Charoy, and C. Godart, "Dynamic authorisation policies for event-based task delegation," in *CAiSE*, 2010, pp. 135–149.