A4.D3.2 – Evaluation of V1 of Dynamic Validation Prototype

C. Kloukinas, K. Mahbub, G. Spanoudakis

Document Number	A4.D3.2
Document Title	Evaluation of V1 of Dynamic Validation Prototype
Version	1.1
Status	Final
Work Package	WP 4.3
Deliverable Type	Report
Contractual Date of Delivery	30 April 2007
Actual Date of Delivery	19 June 2007
Responsible Unit	CUL
Contributors	CUL
Keyword List	Evaluation, dynamic validation
Dissemination level	PU

Change History

Version	Date	Status	Author (Unit)	Description
0.1	15 April 2007	Draft	K. Mahbub (CUL)	Table of contents, indicative section contents
0.2	30 April 2007	Draft	K. Mahbub (CUL)	Initial Evaluation Results
0.3	30 May 2007	Draft	K. Mahbub (CUL)	Results for violation detection delay
0.4	4 June 2007	Draft	K. Mahbub (CUL)	Results for event transmission delay
0.5	8 June 2007	Draft	K. Mahbub (CUL)	Results for event capturing effect on monitored systems
1.0	15 June 2007	Final submitted to the consortium for review	G. Spanoudakis (CUL)	Review and editing
1.1	18 June 2007	Final	K.Androutsopoulos (CUL)	Changes to make the document compliant with the quality plan

Executive Summary

This deliverable presents the results of an initial evaluation of V1 of the dynamic validation tool that has been developed in SERENITY. The evaluation has focused on the performance of monitoring and event capturing and the effect of the latter on the systems which are being monitored. The evaluation was based on a series of simulations which generated random event sequences representing executions of a service based system that had been developed at City University. The experiments showed that on average the detection of a violation is timely with the maximum average delay that was observed for typical S&D properties being in the order of seconds. They also indicated that the number of formulas which are being monitored affects performance but not the size of the domain of the variables in the formulas (at least for the main type of security properties that we expect to monitor at runtime). Experiments have also shown that the incorporation of assumptions in the monitoring process and past formulas did not have a significant effect on performance. The evaluation has also been concerned with the communication costs of transmitting events from event captors to the dynamic validation tool and has indicated that when both these components are running on the same machine the relevant cost is low. Event capturing, however, has been found to have a significant effect on the performance of the system that is being monitored. As part of the initial evaluation that we carried out we introduced certain optimisations in the monitoring engine of the dynamic validation tool. These optimisations will be released with the second version of the tool. Further optimisations are also under investigation for incorporation in the next version of the tool including the implementation of index-based searches of monitoring formula templates during the monitoring process and the development of a distributed version of the dynamic validation tool.

Table of Contents

1. Introduction

1.1. Overview

In this deliverable, we describe the results of an initial evaluation of the dynamic validation prototype of SERENITY (see deliverable A4.D3.1). The main purpose of this evaluation was to inform the further development of the dynamic validation prototype and the mechanisms supporting it (e.g. the supporting information collection mechanisms).

As it had been identified in the deliverable A7.D5.1 [3], the evaluation of the dynamic validation prototype of SERENITY would be concerned on:

— the effectiveness of event capturing mechanisms

— the effectiveness of the monitoring engine

The evaluation of the effectiveness of the monitoring engine focused on assessing the delay in the detection of violations of monitoring rules as the main indicator of the performance of the monitoring engine and investigating different factors that may affect this indicator. These factors included:

- the types of events that should be taken into account for the monitoring of different types of formulas (i.e. events which are captured during the operation of the system that is being monitored and events which are generated by the engine through deduction)
- the effect of the number of the monitored formulas on the delay in the detection of violation, the effect of the size of the domains of the variables used in formulas in the detection delay and
- the effect of interdependencies between the monitored formulas on monitoring performance.

The delay in the detection of violations of monitoring rules was studied in average terms as analytical results that have been reported in [5] have demonstrated that the algorithm which is implemented by the dynamic validation tool developed in SERENITY has an exponential worst case time complexity.

The evaluation of the event capturing mechanisms of SERENITY focused on two areas:

- The effect of event capturing on the performance of the system that is being monitored.
- The delay in the communication of events to the dynamic validation tool.

The initial evaluation activity that we have carried out and is reported in this deliverable has not been concerned with other possible criteria of evaluation that had been identified in [3], including tool usability, the expressiveness of the monitoring language and usability of the monitoring language for specifying monitorable properties. The assessment of these indicators is to be based on input from the specifiers of the patterns in the project and therefore would have to be delayed until the main part of the pattern specification activity is completed in the project in order to have a more thorough basis for assessing the ability of the language that the monitoring offers for specifying monitoring rules. Furthermore, the expressive power of event calculus that underpins the monitoring language of the dynamic validation tool has been demonstrated in the literature and, therefore, reassessing it in the context of SERENITY was not a priority. Also the general usability of the dynamic validation prototype should be based on the more advanced second version of the tool.

To enable a controlled study of the factors that were the focus of our initial evaluation, our experimental evaluation was based on simulation of the operation of a service based system and the monitoring of specific forms of security properties during it, namely availability and integrity properties. The focus on these types of properties was due to the fact that they constitute 2 of the 3 basic types of security properties. Confidentiality, which is the third type of the basic security properties, was not used in our experiments. This however does not affect the generality of our results since, as we have discussed in [7], confidentiality properties can be expressed in the language of the dynamic validation prototype by monitoring rules and assumptions which have the same general form as the monitoring rules and assumptions that express integrity properties.

1.2. Report Structure

The rest of this report is structured as follows.

In Section 2, we give an overview of the setting of our experiments describing the criteria that we used during the evaluation, the system that subjected to monitoring during the evaluation, the way in which we simulated the operation of this system to perform monitoring, and the different sets of rules that we used during the monitoring process.

In Section 3, we present and discuss the main results of our evaluation and give an overview of some optimisations that we introduced to the dynamic validation prototype in order to address some issues that were identified early in the evaluation process.

In Section 4, we present an overview of the main findings of our experiments and the identified needs for improvements in the dynamic validation prototype.

Finally, in Section 5, we provide some concluding remarks for our evaluation.

2. Experimental Setup

2.1. Criteria of Evaluation & Relevant Measures

As we discussed in Section 1, the main aim of our experiments was to evaluate the performance of the monitor an event captors.

The evaluation of the performance of the monitor was based on the following criteria:

— The average time delay in detecting a formula violation and the effects on this measure of the following factors:

— The use of formulas with interdependencies (i.e., monitoring rules and assumptions that can generate information required by the rules) as the monitoring of these formulas requires both recorded and derived events and therefore it engages the deductive reasoning capabilities of the monitor.

— The number of monitored formulas (rules and assumptions).

— The size of the domains of the non time variables of monitored formulas.

— The average time that elapses before an event that is received by the monitor is processed by it.

The evaluation of event captors was based on the following criteria:

— The delay in the transmission of events from event captors to the monitor

— The effect of event capturing in the performance of the system that is being monitored

Also, as we discussed in Section 1, to enable a controlled study of the above factors, our experimental evaluation was based on simulation of the operation of a service based system This simulation generated events that were used in the monitoring sessions that we executed during the evaluation.

The basic measures that we used in order to evaluate the performance of the dynamic validation tool against the above criteria are defined below. It should be noted that in general the dynamic validation tool and the system that is being monitored by it (regardless of whether it is based on a really executed or a simulated BPEL process) are two different processes and therefore they have different clocks. Consequently, it is necessary to translate the timeline of the two processes into a common timeline and take measurements from it. To achieve this, in our experiments we transformed the timestamps of the events of the system that is being monitored into the time line of the monitor. Given this time translation principle, Table 1 shows the definitions of the basic time measures that we used in our experiments.

Time	Meaning/Calculation
t_i^e	This is the time of occurrence of an event i. This is the time when the event is captured from the system that is being monitored or is generated by the simulator.
t_s^m	This is the starting time of the monitor.

Time	Meaning/Calculation
t_c^m	This is the current time of the monitor.
$t_i^{e(d)}$	This is the time when an event i is recorded in the monitor's event database. $t_i^{e(d)}$ is computed by the formula $t_i^{e(d)} = (t_i^e - t_0^e) + t_s^m$ where t_0^e is the time of the occurrence of the first event that is used in a monitoring session.
t_i^m	This is the time when the monitor retrieves an event i from its event database to process it.
t_s^{Fj}	Starting time of the decision procedure that the monitor executes to check for violations given the truth values of the predicates in the template j of a formula F
t_e^{Fj}	This is the completion time of the decision procedure that the monitor executes to check for violations given the truth values of the predicates in the template j of a formula F

Table 1 – *Basic Time Measures*

Given the basic time measures shown in Table 1, we define the following performance measures:

— Average waiting time of an event (e-delay):

The waiting time of an event is the difference between the time when the event is stored in the event database of the dynamic validation tool and the time when the event is retrieved from this database by the monitor to be processed.

The average waiting time of events, called *e-delay*, is measured using the following formula,

$$\text{e-delay} = \sum_{i=1,\ldots,K \text{ where } t_i^m - t_i^{e(d)} > 0} (t_i^m - t_i^{e(d)}) \; / \; K$$

where K is the total number of events. Figure 1 illustrates the waiting time of an event.

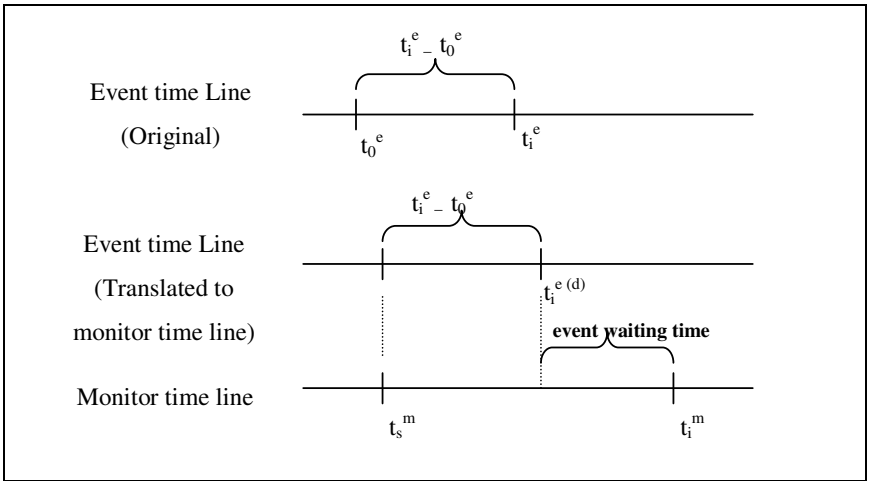


Figure 1 – Event waiting time

— Average decision delay:

The delay in making a decision about a possible violation (or satisfaction) of a rule instance is measured as the difference between the time when the monitor makes a decision for the rule instance and the time when the last event e_i that makes it possible to make this decision was recorded in the event database (or the time when the monitor picks up the event e_i from the event database, in cases where the monitor is idle). The event that makes it possible to decide about the satisfaction/violation of a rule instance is the event that is used to update the truth-value of the last predicate in the template.

The average delay in making a decision for a template is measured according to the following formula,

$$\text{d-delay} = \sum_{i=1,\ldots,N} d_j / N$$

where

— N is the number of templates for which a decision has been made

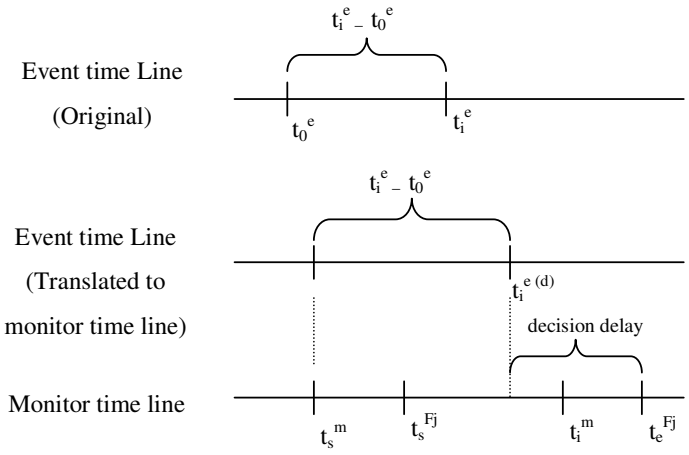
— d_j is the delay in making the decision for template j of the formula F that is computed as

— $d_j = t_e^{Fj} - \max_{i \in Fj}(t_i^{e(d)})$ if $t_e^{Fj} - \max_{i \in Fj}(t_i^{e(d)}) > 0$ (I)

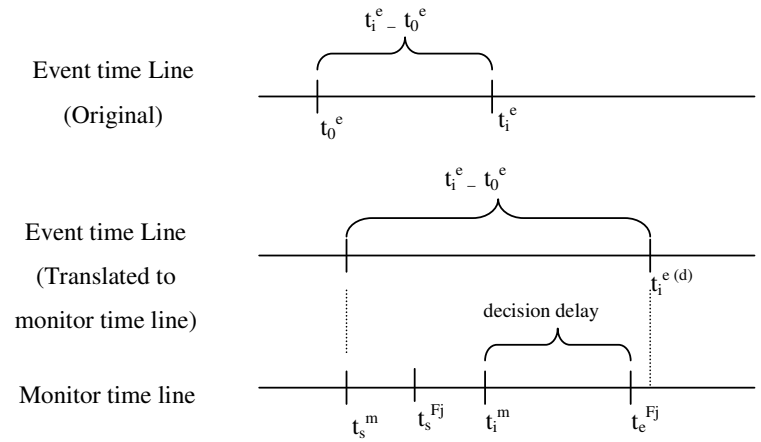
— $d_j = t_e^{Fj} - \max_{i \in Fj}(t_i^{m})$ otherwise (II)

where i ranges over the events used to establish the truth values of the predicates in F_j. Formula (I) above is used to compute the delay in making a decision about a template in cases where the monitor starts checking this template after the occurrence of all the events that were used to instantiate and set the truth values of the predicates in the template. Formula (II) is used in cases where the monitor is capable of checking a template before the last event that was used to instantiate one of its predicates really occurred. This case is only possible due to the use of simulations when the time of the real occurrence of an event could be after its generation by the simulator and its transmission to the monitor. Figure 2 illustrates the two cases where formulas (I) and (II) should be used to compute the decision delay for a particular template instance. In Case A, the monitor completes the decision process after the instantiation of the last predicate in the formula template at time $t_i^{e(d)}$. In Case B, the monitor completes the decision process before the time that

instantiation of the last predicate in the formula template took place. In this case the decision delay is measured as the time that it took the monitor to complete the decision process after retrieving the event that instantiated the last template in the formula from its event database.



Case A: Use of Formula (I)



Case B: Use of Formula (II)

Figure 2 – Decision delay

2.2. The Monitored System and Formulas

To evaluate the dynamic validation prototype we used a *Car Rental System* (CRS) as the system to be monitored. This system 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. CRS is implemented as a service composition workflow that is specified in BPEL [1] and orchestrates *Car Information*

Services (IS), *Customer Management Services* (CMS), *User Interaction Services* (UI) and *Sensoring Services* (SS). These services realise the following functionalities for CRS:

— Car information services (IS services) maintain registries of cars that can be rented, check car availability upon car rental requests and allocate cars to customers as requested by CRS. The design of CRS assumes that IS services are provided by different car rental companies to offer the above functionalities.

— Sensoring services (SS services) detect movements of cars as they are driven in or out of car parks and inform CRS accordingly. SS services are provided by different car parks to

— The Customer Management Service (CMS service) maintains the database of the customers of CRS and authenticates these customers as requested by CRS.

— The User interaction services (UI) provide CRS with different user interfaces that can handle interactions with the end-users on different devices.

In a typical operational scenario, CRS receives car rental requests from UI services, authorises customers contacting CMS and checks for the availability of cars by contacting IS services. It also gets car movement information from SS services. This information is used to track the status of a car. The BPEL specification of the CRS composition process and the WSDL files of the web services deployed by it are presented in Appendix A.

```
Rule ID R1
forall  t1 : time, exists  t2 : time
Happens(ic:makeAvailable(ID,status1,sender1,receiver1,source1,loc,carId),t1,R(t
1,t1)) ∧
Happens(ir:makeAvailable(ID,status2,sender2,receiver2,source2),t2,R(t1,t2))
⇒ oc:self:sub(t2,t1) < 500

Rule ID R2
forall  t1 : time, exists  t2 : time
Happens(ic:isAvailable(ID,status1,sender1,receiver1,source1,loc),t1,R(t1,t1)) ^
Happens(ir:isAvailable(ID,status2,sender2,receiver2,source2,carId),t2,R(t1,t2))
⇒ oc:self:sub(t2,t1) < 500
```

Figure 3 – *Set-1* of monitoring rules

In our experiments, we used the BPEL specification of CRS to generate events simulating its actual behaviour at runtime. This was done with the use of a *simulator* that we describe in Section 2.3 below. The simulator was used to generate sequences of random events representing the execution of different complete paths in the workflow of CRS. Further details on the generation of these sequences are given in Section 2.3. The use of simulation was necessary in order to be able to generate sets of events with varying characteristics (e.g. varying sizes of the domains of the variables of monitored rules) and study their effect on the performance of the monitor, something that would be difficult with a real execution of the CRS workflow.

In our experiments we monitored two separate sets of monitoring rules and assumptions which are shown in Figure 3 and Figure 4.

The first set of rules (referred to as "Set-1" henceforth) consisted of two monitoring rules R1 and R2, checking a bounded form of the availability of the operations *makeAvailable* and *isAvailable* of the CRS BPEL process, respectively. As shown in Figure 3, these rules express a bounded form of availability in which availability is defined as the ability of the relevant operations to respond within 500 milliseconds after they are called following the pattern discussed in [7].

Assumption ID A1

```
forall  t1 : time
Happens(ic:returnKey(ID,status,sender,receiver,source,carId,loc),t1,R(t1,t1))⇒
Initiates(ic:returnKey(ID,status,sender,receiver,source,carId,loc),
authorised_availability(carId),t1)
```

Assumption ID A2

```
forall  t1 : time, exists t2 : time
Happens(ic:makeUnAvailable(ID,status,sender,receiver,source,carId,loc,custId),t
1,R(t1,t1)) ∧
HoldsAt(authorised_availability(carId),t1) ⇒
Terminates(
ic:makeUnAvailable(ID,status,sender,receiver,source,carId,loc,custId),
authorised_availability(carId),t2) ^ t2 >= t1+1 ^ t2 <= t1+1
```

Rule ID R3

```
forall  t1 : time
Happens(ic:makeAvailable(ID,status,sender,receiver,source,carId,loc),t1,R(t1,t1
)) ⇒
HoldsAt(authorised_availability(carId),t1)
```

Figure 4 – *Set-2* of Monitoring Rules

The formulas shown in Figure 4 constituted the second set of monitoring rules (referred to as "Set-2" in the following). These formulas expressed one monitoring rule (R3) and two assumptions (A1 and A2) that were used to deduce information about the state of CRS during a monitoring session. More specifically, assumption *A1* in Figure 3 specifies the initiation of a fluent that signifies the authorisation of the availability of a car when the key of this car is returned to a car park (see the operation `ic:returnKey(ID,status,sender,receiver,source,carId,loc)`).Assumption *A2* is used to terminate the fluent that signifies the availability of a car whenever the operation `makeUnAvailable` of the *IS* service is invoked to make this car unavailable. Finally, *R3* is used to check whether in the cases where the *IS* marks a car as available, a fluent that authorises the

availability of the car holds (i.e., exists in the fluent database that is maintained by the monitor; see [2] for more details).

The two sets of formulas presented above were selected in order to introduce certain forms of variability in the experimentation process to enable the investigation of the performance of the monitor with respect to some of the criteria introduced in Section 2.1. More specifically, the second set of rules included formulas with interdependencies. The predicate **HoldsAt**`(authorised_availability(carId),t1)` in R3, for instance, depends on[1]

— the predicate

Initiates`(ic:returnKey(ID,status,sender,receiver,source,carId,loc), authorised_availability(carId),t1)` of assumption A1, and

— the predicate

Terminates`(ic:makeUnAvailable(ID,status,sender,receiver,source,carId,loc,custId), authorised_availability(carId),t2)` of assumption A2.

Thus, checking the formulas in *Set-2* requires both recorded events and events that are derived from recorded events using the assumptions of the set and the deduction process of the monitor. Unlike it, the formulas in *Set-1* can be checked by taking into account only the events which are recorded and reported to the monitor by event captors.

2.3. The Simulator

In our experimental evaluation we used event sequences that were generated by simulating the BPEL process described in Section 2.2 using a simulator developed at City University that is described in [5]. This simulator gets as input

(i) A set of EC-Assertion formulas that represent the different complete execution paths than exist within the BPEL process that is to be simulated (this set is generated automatically from the BPEL process by one of the supporting tools of the simulator).

(ii) The size n_v of the domain of each of the non time variables v which appear in the formulas in (i) above. These variables correspond to the input and output parameters of the service operations which appear in the BPEL process that is to be simulated.

(iii) A function, called $dist_{exec}$, which determines the distribution of the values of the time that elapses between the executions of two complete consecutive paths of the BPEL process. This function indicates the time that elapses between the initiations of two different transactions in the BPEL process.

(iv) A function, called $dist_{open}$, that determines the distribution of the occurrence time of the constrained predicates in a formula (i.e. execution path) for which one of the lower boundary (LB) or the upper boundary (UB) has not been specified.

(v) A number M that indicates the number of events that should be generated by simulating the process.

[1] This dependency arises by virtue of the Event Calculus axioms (see Table 1 in [2]).

Given (i)–(v), the simulator generates a set of n_v random values for each variable v in the BPEL process that constitutes the *domain* of the variable in the simulation (n_v is the size of the domain of v that is passed as input to the simulator) and then generates a set of M events for the process using the following procedure.

— It selects randomly a formula, Fi, from the set of the EC-Assertion formulas that represent the different execution paths in the BPEL process and generates random events from the formula. These events preserve the temporal order which is imposed by the time constraints of the formula and are generated as follows.

- Initially, the time stamp (ST_i) of the first event of the formula which always corresponds to the unconstrained predicate in F_i is determined. ST_i is computed using the formula

  $$ST_i = ST_{i-1} + offset_i$$

  where $offset_i$ is the random value generated from the distribution function $dist_{exec}$ and ST_{i-1} is the starting time of the formula that had been simulated in the previous step (ST_0 is the starting point of the whole simulation).

- The timestamp of each of the successive events which can be generated from F_i is computed based on the lower boundary (LB) and upper boundary (UB) of the time variable of the predicate. More specifically, if the values of both LB and UB of the next predicate in F_i from which an event is to be generated are known, the simulator generates a random timestamp in the range [LB, …, UB] using the uniform distribution function $dist_{uni}$ in this range. In cases, however, where LB or UB are not known then the timestamp of the predicate is determined by the following formulas:

  $Timestamp = LB + offset$, if LB is known

  $Timestamp = UB - offset$, if UB is known

  where *offset* is the random number that is generated by applying the distribution function $dist_{open.}$

- When the *simulator* generates an event for a predicate with a time stamp, it also updates the upper boundary (UB) and lower boundary (LB) of ranges of all the other predicates in the formula that depend on the time variable of this predicate. Also, for the non time variables of the predicate used to generate the event that have not been already assigned some value, the *simulator* selects randomly a value from the domain of the respective variable. The selection of a random value from the domain assumes that different values in the domain have equal probability (i.e. each value has a probability 1/n, where n is the size of the domain) of being selected.

- The *simulator* also assigns a unique *id* to each generated event. All the events which are generated in a simulation must have unique *id*s with the exception of the following cases,

  — Events that instantiate pairs of predicates in a formula that signify a receive activity and the corresponding reply activity. Such events must have same id.

  — Events that instantiate pairs of predicates in a formula that signify an invocation of an operation of an external web service and the corresponding response from that web service. These events are assigned the same id in order to be able to correlate the invocation with the response.

— Events generated as instances of a predicate that signifies a receive activity in a formula and the predicates in the same formula that signify the initiation of fluents to represent the value of the input variable of the operation called by the receive activity. These events must have same id in order to be possible to correlate them.

— Events generated as instances of a predicate that signifies response from the execution of an operation in an external web service and all the predicates in the same formula that signify the initiation of fluents to represent the value of the output variable of the relevant operation. These events must have same id in order to be possible to correlate them.

— After generating events from F_i, the simulator selects another formula that represents a complete execution path in the BPEL process and generates events from it using the above procedure. This process continues as many times as it is required in order to generate the required number of events.

Example

As an example of the BPEL simulation process that is described above, consider a simple BPEL process whose execution paths are represented by the formulas *F1* and *F2* in Figure 5. Assuming that, the size of the domains of the variables _x, _y, and _z in these formulas are set to 4, 5, and 5, respectively, the *simulator* will generate the following domains for the variables:

— domain of *_x* : {aqa, yab, vsbac, zpad}

— domain of *_y* : {348, 9856, 3401, 23, 65923}

— domain of *_z* : {btma, nfbrc, hbwqd, mbsqe, djbfk}

Then, assuming that

— The mean and variance of the distribution function $dist_{exec}$ are 0.8 seconds and 0.2 seconds respectively.

— The mean and variance of the distribution function $dist_{open}$ are 0.2 seconds and 0.5 seconds respectively.

the simulator will generate a random initial timestamp ST_0= *1135694663208*.. Subsequently assuming that *F2* is the first randomly selected formula (execution path), the following events will be generated from it.

Event 1: The first event to be generated from *F2* will correspond to the unconstrained predicate, **Happens**(in:p:A(_ID1,_x),t1,R(t1,t1)) in the formula. This event signifies an invocation of an operation called A in the external web service *p*. The simulator will randomly pick a value for variable *_x* from its domain, say *yab*, and assign a unique *id* to the first event. The time stamp for this event will be ST_0. Thus, the first event that will be generated from *F2* will be:

Happens(in:p:A(id1,yab), *1135694663208*)

```
(F1) forall t1:time, _z:string
     Happens(rc:s:O(_ID),t1,R(t1,t1)) ∧
     Initiates(rc:s:O(_ID),valueOf(z,_z),t1) ∧ (∃ t2)
```

```
        Happens(re:s:O(_ID,_z),t2,R(t1,t1 + tu * 100))

(F2)  forall t1:time, _x:string, _y:int
        Happens(in:p:A(_ID1,_x),t1,R(t1,t1)) ∧ (∃ t2)
        Happens(ir:p:A(_ID1),t2,R(t1,t2)) ∧
        Initiates(ir:p:A(_ID1),valueOf(y,_y),t2) ∧ (∃ t3)
        Happens(in:q:B(_ID2,_x,_y),t3,R(t2,t2 + tu * 50))

** tu = 1 ms.
```

Figure 5 – Execution paths of a BPEL process expressed as EC formulas

Event 2: The second event to be generated from F2 corresponds to the second predicate in F2, `Happens(ir:p:A(_ID1),t2,R(t1,t2))` which signifies the response from the execution of the operation A in the external web service that was invoked by `Happens(in:p:A(id1,yab), 1135694663208)`. The value of the lower boundary t1 of the second event is set to 1135694663208 (i.e., the same as the time stamp of event 1) and the value of the upper boundary t2 is undefined. The simulator uses the distribution function $dist_{open}$ to compute a random number which it subsequently adds to t1 to determine the value of t2. Thus, if the computed random number is 0.004 the simulator converts this number into milliseconds ($dist_{open}$ is specified in seconds but the simulator's clock operates in milliseconds as t_u=1 ms) and adds it to t1 to obtain the value of t2, i.e. t2 = 1135694663212. Furthermore, the ID of the event generated for this predicate must be the same as the ID of the event 1.Thus, the second event generated from F2 will be:

```
Happens(ir:p:A(id1), 1135694663212)
```

Event 3: The third event to be generated from *F2* corresponds to the third predicate in the formula, i.e. `Initiates(ir:p:A(_ID1),valueOf(y,_y),t2)`. This predicate signifies the initiation of fluent due to the response from an external web service. The second predicate in *F2* signifies the response from the external web service that initiates this fluent. Therefore the ID of the event generated for the third predicate must be the same as the ID of the *event 2*. The value of t2 is *1135694663212* (due to the *event 2* time stamp). The *simulator* randomly picks a value for the variable *_y* from its domain, let it picks 3401. The third event generated from *F2* will be,

```
Initiates(ir:p:A(id1),valueOf(y,3401), 1135694663212)
```

Event 4: The fourth event to be generated from the F2 corresponds to the fourth predicate in the formula, i.e. `Happens(in:q:B(_ID2,_x,_y),t3,R(t2,t2 + tu * 50))`. This predicate signifies the invocation of an operation in an external web service. By virtue of the generation of event 3, the variable _x has been assigned to the value yab (due to event 1) and the variable _y has been assigned to the value 3401 (to event 3). The value of the lower boundary of the time variable of the predicate is 1135694663212 as it must be the same as the time stamp of event 2 and the value of the upper boundary of the time variable of the predicate is 1135694663262 as it must be equal to the timestamp of event 2 plus 50 time units tu. After establishing the range of t3 as

$\Re$(1135694663212, 1135694663262), the simulator creates a random number in this range according to $dist_{uni}$ and assigns it to t3. Assuming that the computed random number is 1135694663236, the fourth event generated from F2 will be:

```
Happens(in:q:B(id2,yab,3401), 1135694663236)
```

When the simulator generates the events for all the predicates in F2, it will randomly select another formula from the formulas of Figure 5, say F1. To determine the time stamp of the first event generated from F1 the simulator uses the distribution function $dist_{exec}$ to compute a random number and update the value of ST by adding the random number to the previous value of ST. Assuming that the computed random number is 1.35, the simulator converts this number into milliseconds ($dist_{exec}$ is specified in seconds but the simulator's clock operates in millisecond as tu=1 ms) and adds it to the previous value of ST to obtain the new value of ST, i.e. ST = 1135694664558. Hence the time stamp of the first event generated from F1 (due to the unconstrained predicate) is 1135694664558. The subsequent events from F1 are generated from F1 following the same mechanisms as in case of F2 described above. The simulator repeats this event generation steps for as long as the simulation of a BPEL process takes place.

2.4. The Deployed Machines

In our experiments we used two machines:

— A Windows XP machine with an Intel Pentium at 3.00 GHZ and 2GB of RAM

— A Windows XP machine with an Intel Pentium at 3.20 GHZ and 1GB of RAM

The former machine was used in all the experiments regarding the performance of the monitoring engine of the dynamic validation tool (see Experiments 1-5 below) and the latter was used in the experiments regarding the event captors (see Experiments 6-8 below).

3. Results

3.1. Violation Detection Delay

(i) Experiment 1: Monitoring Set-1 with Initial Version of the Dynamic Validation Prototype

In the initial experiment, we measured the average and standard deviation of the decision delay whilst monitoring the formulas in *Set-1* of Figure 3. The monitoring session was based on a set of 20,000 events that was generated by simulating the CRS BPEL process shown in Appendix A. The used set of events corresponded to a total of 2,981 transactions (i.e., executions of complete CRS process paths) taking place over a period of 16.527 hours. This was the equivalent of 180.37 transactions per hour. The set of events used in this experiment was generated by using the following random time value distributions:

— $dist_{exec}$ had been set to the normal distribution with mean and variance values set at 0.8 seconds and 0.2 seconds, respectively.

— $dist_{open}$ had been set to the normal distribution with mean and variance values set at 0.2 seconds and 0.5 seconds respectively.

Furthermore, in the simulation, domain size of the non time variables of the formulas that take as values cars (i.e., variable `carId`) and car parks (i.e., variable `loc`) was set to 20 cars and 3 car parks.

#Events	Average D-Delay (secs)	Standard Deviation of D-Delay (secs)	#Templates	Average Event Waiting Time (secs)
2000	1.099	0.639	971	0
4000	2.162	1.259	1991	0
6000	3.373	2.195	2988	0
8000	4.682	3.092	3996	0
10000	5.909	3.773	4989	0
12000	7.208	4.649	5967	0
14000	18.887	92.033	6969	10.21
16000	1008.706	2956.387	7982	988.238
18000	3481.446	7618.506	9003	3394.419
20000	7126.057	13214.251	9999	7038.411

Table 2 – Monitoring Statistics for *Set-1*

Table 2 shows the average and standard deviation of the decision delay (d-delay) as measured for every 2,000 events in the monitoring process, the number of formula templates that had been created to represent different formula instances, and the average event waiting time at these points in the monitoring process. Figure 6 shows a graph of the average decision delay and the number of rule violations that were found during monitoring and Figure 7 shows a graph of the average event waiting time at the different stages of the monitoring process in the experiment. All time measures in Table 2 and these figures are given in seconds.

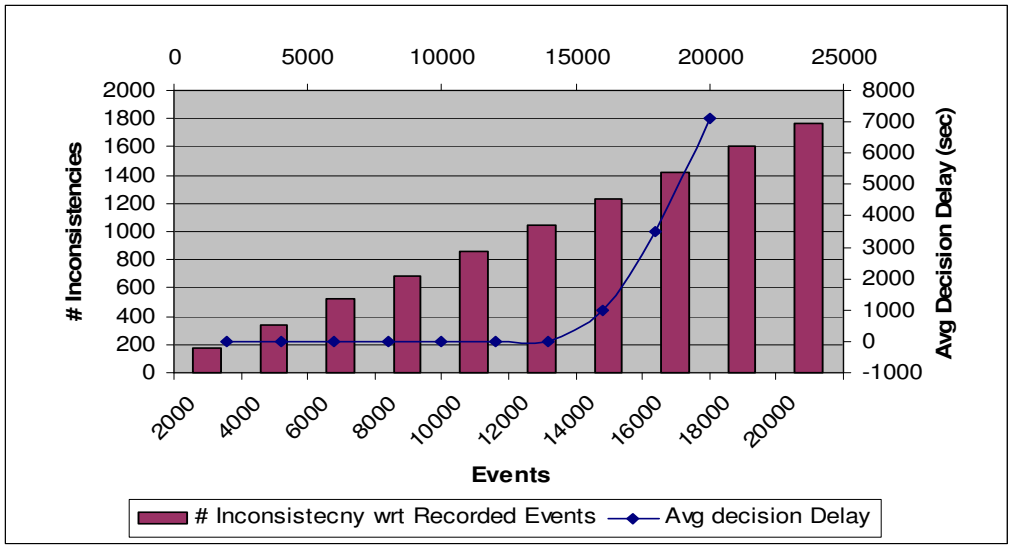


Figure 6 – Average decision delay and number of inconsistencies with respect to recorded events in Set-1

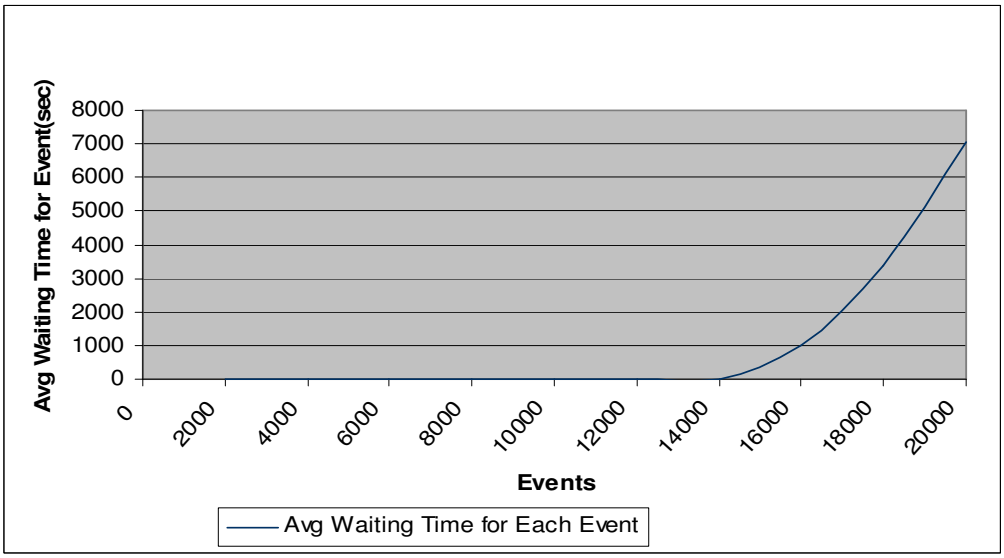


Figure 7 – Average waiting time for each event for Set-1

As shown in the graph in Figure 6, the number of rule violations grew linearly during the simulation (as expected due to the controlled nature of the experiment) the average decision delay did not change up to 12,000 events and then it started to rise exponentially.

This is due to the algorithm used in the implementation of the dynamic validation tool (see [5] for a formal specification and analysis of this algorithm). At runtime, the dynamic validation prototype maintains templates that represent different instantiations of the formulas to be monitored. In these templates the variables of the formulas (or a subset of them) are unified with specific values. The dynamic validation prototype picks events from the event database and checks if there are instances of templates that should be updated by the events. Updates may be made if the signature, the event variable bindings and the time of the event comply with the predicate signature, the current predicate variable bindings, and the time range of the predicate in a template, respectively. If a predicate is updated, the bindings of the predicate's variables in the template are also updated. New instances of templates are generated if the event corresponds to an unconstrained predicate of a template (i.e., a predicate whose time variable is not constrained by the time variable of another predicate), or the variable bindings of the predicate have values that are different from the event variable bindings values. The truth-value of a predicate in a template instance may also be updated by applying the principle negation as failure.

The exponential rise of the average decision delay occurred because for each event the monitor has to check if it can be unified with each of the template instances. The same effect is reflected in Figure 7. As shown this figure, the waiting time for each event is negligible (0 milliseconds) up to 12,000 events but after that point it rises sharply.

Although the results of the initial experiment indicated that the support for typed variables in monitoring rules and the other extensions of the original monitoring engine described in [5] that were developed in SERENITY did not have a significant effect in the time that is required for the detection of violations, the exponential rise in the average time that was required to detect a violation in the SERENITY engine was a concern that prompted an investigation for possible optimisations.

(ii) Template pruning

The main area that we looked at was how to reduce the increase in the number of active templates during the monitoring process. This investigation focused at the process of template creation and the possibility of pruning active templates which do not provide sufficient information for making a decision about the rule instance that they represent and cannot be possibly updated by further events. In the monitoring scheme implemented by the monitor, a new instance of a template is created if a predicate has variables which are bound to values that are different from a not processed yet event that could be unified with the template. This may create many template instances that are not needed in monitoring.

For example, consider rule R2 in Figure 3. When an event that represents the invocation of the operation *isAvailable*, with unique ID say *isAvail120*, is encountered the monitor creates a new template instance and unifies the event with the predicate `Happens(ic:isAvailable(ID,status1,sender1,receiver1,source1,loc),t1,R(t1,t1))` in the template. Subsequently, when an event that represents the response from the particular invocation of the operation *isAvailable* is encountered, the monitor unifies it with the predicate `Happens(ir:isAvailable(ID,status2,sender2,receiver2,source2,carId),t2,R(t1,t2))`

in the template. However, to cover the possibility of having another response from this operation with different variable values, the monitor was also creating a copy of this template. Although this functionality was necessary in order to ensure the completeness of the reasoning process implemented by the monitor, in this particular example the creation of the template copy was not necessary. This was due to the fact that due to the semantics of the operation invocation in the particular example, there could only be one response for the call. To address this issue we amended the template creation process in the case of unification of templates with events that represent responses from operation invocations and re-executed the experiment. This optimisation led to dramatic improvement in the monitoring results which are presented in the following.

(iii) Experiment 2: Monitoring Set-1 with Optimised Implementation (OI)

Following the optimisation in the unification process that we described above we re-executed the experiment with the formulas in Set-1 of Figure 3 using the same set of events that was used in (i) of Section 3.1.

Table 3 shows the average and standard deviation of the decision delay (d-delay) as measured for every 2,000 events in the monitoring process, and the number of formula templates that had been created to represent different formula instances, and the average event waiting time at these points in the process. Also, Figure 8 shows a graph of the rule violations (inconsistencies) and the average decision delay at the different stages of the monitoring process in the experiment. All times measures in this figure and table are in seconds.

#Events	Average D-Delay (s)	Standard Deviation of D-Delay (s)	#Templates	Average Event Waiting Time (s)
2000	0.385	0.189	297	0
4000	0.756	0.431	600	0
6000	1.128	0.657	907	0
8000	1.478	0.865	1202	0
10000	1.841	1.081	1509	0
12000	2.188	1.300	1794	0
14000	2.566	1.555	2084	0
16000	2.916	1.772	2360	0
18000	3.382	2.191	2661	0
20000	3.887	2.651	2955	0

Table 3 – Monitoring Results for Set-1 (OI)

As it is evident from these results the template pruning optimization that we discussed in (ii) above improved the performance of the monitoring process dramatically (a 4-orders of magnitude decrease in the average decision delay was recorded for 20,000 events). Also, the average decision delay appeared to be increasing linearly with the number of events indicating a very reasonable performance over a monitoring session of a significant length and number of executed transactions (16.527 hours and 2,981 transactions as we discussed in (i) above). These results are certainly related to the particular form of the monitored formula that enabled the effective pruning of templates. As we have, however, discussed in [7], the formulas that can be used to monitor basic security properties such as availability, integrity and confidentiality either have the same basic form as the one used in this experiment (availability) or have forms which cannot lead to the generation of unused templates in the monitoring process (confidentiality and integrity). This point about was confirmed by the results of the experiment with *Set-2*. In this experiment we used an integrity formula which has the same general structure as confidentiality formulas (see [7]). The spare capacity of the monitor is also indicated by the fact that the average event waiting time within the region of 18,000 to 20,000 events remained at 0 seconds (see Table 3).

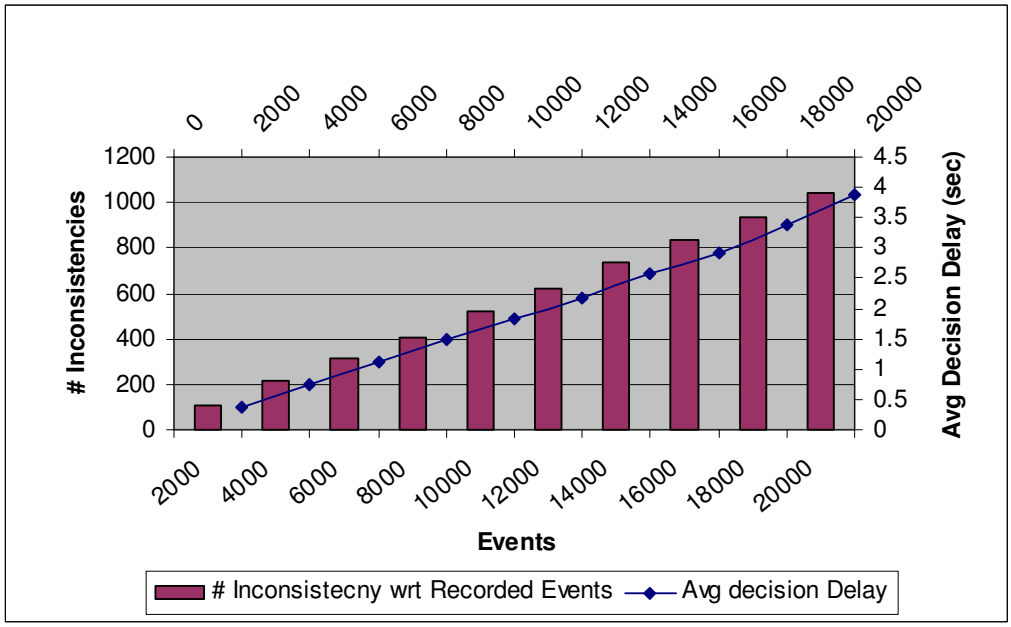


Figure 8 – Average decision delay and number of inconsistencies for *Set-1* (OI)

(iv) Experiment 3: Monitoring Set-2 with Optimised Implementation (OI)

In the third experiment, we used the formulas in *Set-2* of Figure 4. The monitoring rule R1 in this set is used to check the integrity of the execution of the operation `makeAvailable` in CRS. The assumptions A1 and A2 in the set are used to generate fluents and store them in the fluent database of the monitor. The difference between *Set-1* and *Set-2* is that the monitoring of the formulas in the latter set does not only depend on the events captured during the operation of the monitored system, as it is the case with *Set-1*, but also requires searches in the fluent database of the monitor that keeps information about the fluent initialisation and termination events. This is necessary in order to

establish whether the *HoldsAt* predicate in rule R3 of *Set-2* is true or false at specific time points. Furthermore, the monitoring of R3 engages the deduction process of the monitor since this process is necessary in order to generate fluent initiation and termination facts from formulas A1 and A2 and store them in the fluent database of the monitor.

The monitoring of *Set-2* was based on the same set of events that was used in the previous experiments. Table 4 shows the average and standard deviation of the decision delay (d-delay) as measured for every 2,000 events in the monitoring process of this experiment, and the number of formula templates that had been created to represent different formula instances and the average event waiting time at these points in the process. Also, Figure 9 shows a graph of the average decision delay at the different stages of the monitoring process in the experiment. All times measures in this figure and table are in seconds.

#Events	Average D-Delay (s)	Standard Deviation of D-Delay (s)	#Templates	Average Event Waiting Time (s)
2000	0.0788	0.0248	141	0
4000	0.119	0.0465	296	0
6000	0.161	0.070	445	0
8000	0.204	0.095	596	0
10000	0.254	0.1331	744	0
12000	0.302	0.168	885	0
14000	0.353	0.198	1034	0
16000	0.394	0.216	1175	0
18000	0.437	0.237	1322	0
20000	0.481	0.262	1460	0

Table 4 – Monitoring Results for Set-2 (OI)

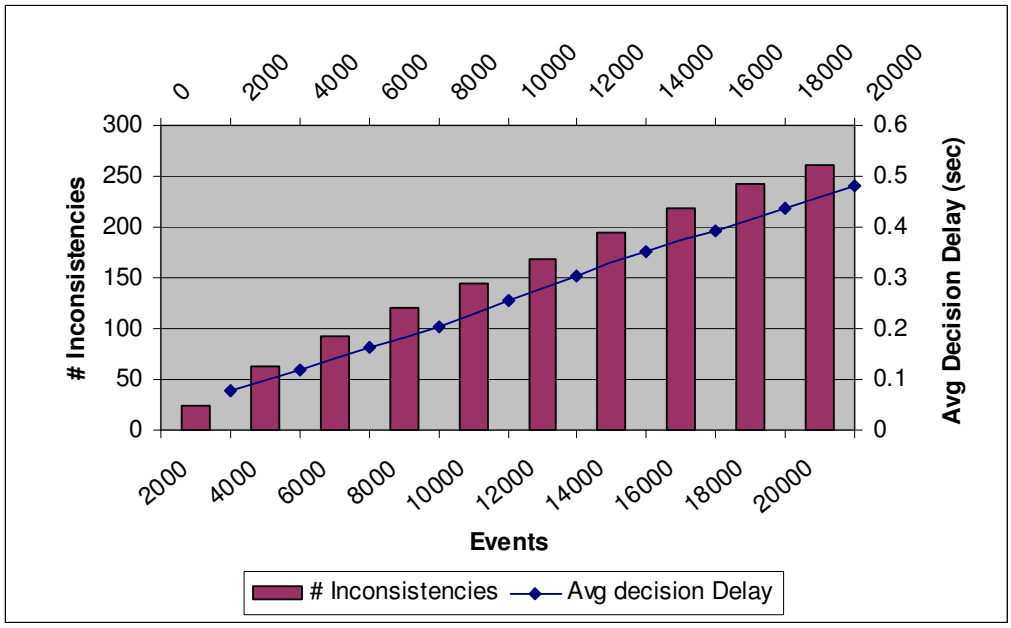


Figure 9 – Average decision delay and number of inconsistencies for *Set-2* (OI)

The results of this experiment were similar to those of the second experiment with the formulas in *Set-1*, as the average decision delay increased linearly with the number of events and remained very low throughout the monitoring process reaching a maximum value of about 0.48 seconds at 20,000 events. This result demonstrated a good performance of the monitor even in this case where the deduction and database search capabilities of it were used. Furthermore, as in experiment 2, the monitor had significant spare capacity as indicated by average waiting time of events that was 0 at the range of 18,001 to 20,000 events (see relevant column of Table 4).

(v) Experiment 4: Investigation of the Effect of Number of Formulas

To explore the effects of number of formulas in the performance of the monitor, we used two variants of *Set-2*. In the first of these variants, referred to as *Set-2'* in the following, we added two more assumptions and one monitoring rule which are listed in Figure 10. These formulas specify another integrity property for the CRS system regarding the operation `departs`. The second variant was created from Set-2' by adding to it another two assumptions and one more monitoring rule.

```
Assumption ID A3
forall  t1 : time
Happens(ic:makeUnAvailable(ID,status,sender,receiver,source,carId,loc,
custId),t1,R(t1,t1))⇒
Initiates(ic: makeUnAvailable
(ID,status,sender,receiver,source,carId,loc,custId),
authorised_depart(carId),t1)
```

```
Assumption ID A4
forall  t1 : time, exists t2 : time
Happens(ic:makeAvailable(ID,status,sender,receiver,source,carId,loc),t1,R(t1,t1)
) ∧
HoldsAt(authorised_depart(carId),t1)
⇒
Terminates(ic:makeAvailable(ID,status,sender,receiver,source,carId,loc),authoris
ed_depart(carId),t2) ^ t2 >= t1+1 ^ t2 <= t1+1

Rule ID R4
forall  t1 : time
Happens(ic:depart(ID,status,sender,receiver,source,carId,loc),t1,R(t1,t1))
⇒
HoldsAt(authorised_depart(carId),t1)
```

Figure 10 – New formulas in *Set-2'*

In the experiment, we used the same set of events that was used in Experiments 1-3. Table 5 summarises the measures taken whilst monitoring each of the three sets Set-2, Set-2' and Set-3' including the average decision delay, standard deviation of decision delay, number of templates and average event waiting time. Also Figure 11 shows graphically the average decision delay in each of three sets (Set-2, Set-2' and Set-2'' are identified as case (1), (2) and (3) respectively in Table 5 and Figure 11).

#Events	Average D-Delay (s)			St. Dev. of D-Delay (s)			#Templates			Avg. Event Waiting Time (s)
	(1)	(2)	(3)	(1)	(2)	(3)	(1)	(2)	(3)	(1), (2), (3)
2000	0.079	0.105	0.171	0.025	0.054	0.107	141	281	517	0
4000	0.120	0.169	0.307	0.047	0.113	0.231	296	569	1029	0
6000	0.161	0.249	0.458	0.070	0.203	0.365	445	839	1541	0
8000	0.204	0.336	0.608	0.095	0.287	0.494	596	1127	2069	0
10000	0.254	0.411	0.761	0.133	0.357	0.635	744	1401	2574	0
12000	0.303	0.492	0.921	0.168	0.434	0.776	885	1680	3090	0
14000	0.353	0.570	1.073	0.198	0.495	0.913	1034	1983	3627	0
16000	0.394	0.646	1.235	0.216	0.565	1.057	1175	2267	4147	0
18000	0.437	0.722	1.421	0.237	0.639	1.254	1322	2553	4673	0

#Events	Average D-Delay (s)			St. Dev. of D-Delay (s)			#Templates			Avg. Event Waiting Time (s)
20000	0.481	0.792	1.611	0.262	0.697	1.470	1460	2841	5181	0

Table 5 – Monitoring Measures for Set-2, Set-2' and Set-2''

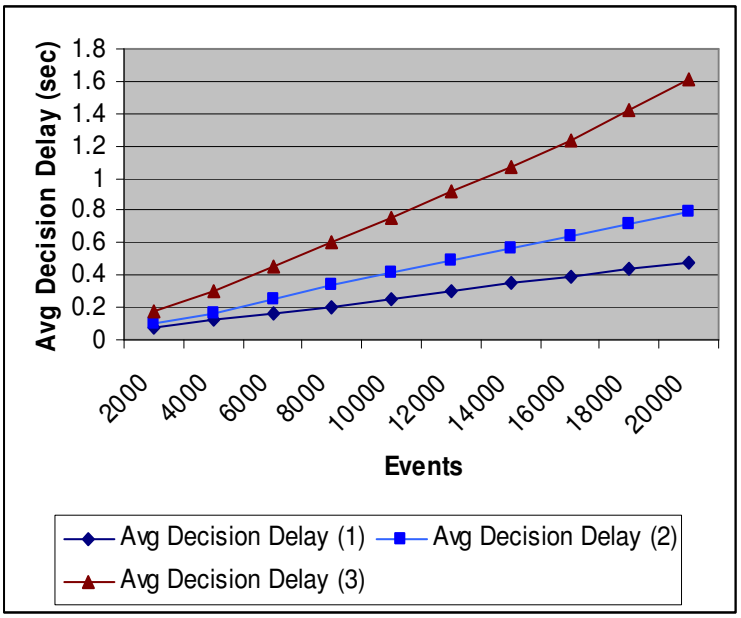


Figure 11 – Graph of Average Decision Delay in *Set-2, Set-2'* and *Set-2''*

As the results of Experiment 5 indicated the average decision delay increased linearly in the case of Set-2' and Set-2'' as it had in the case of Set-2 (see Figure 11) and even in the case of the larger set (Set-2'') it remained at a relatively low level even in the region of 18,000 to 20,000 events (1.611 seconds). The increase in the number of formulas, however, led to an exponential increase in the average decision delay in all event regions. Figure 11 shows this effect for all the event regions as the distance between line (3) and line (2) is larger than the distance between line (2) and line (1) for all event regions and increases along with the number of events. This result indicates that there is a limit of the number of rules/assumptions that can be allocated to a single monitor to check beyond which the average delay in the detection of a violation won't be acceptable, and a distribution of different formulas to different monitors will be required to achieve an acceptable average performance. This issue is further discussed in Section 4.

(vi) Experiment 5: Investigation of the Effect of Size of Variable Domains

In our fifth experiment, we investigated the effect of the size of the domains of the non time variables in the monitored rules on the performance of the dynamic validation prototype. For this experiment we generated a new set of 20000 events for the CRS system and monitored the formulas

in *Set-2'*. This set of events was generated by increasing the size of the domains of the non time variables of the monitored formulas 4 times. More specifically, the domain size of the non time variables of the formulas that take as values customers (i.e., variable `custID`), cars (i.e., variable `carID`) and car parks (i.e., variable `loc`) were set to 200 customers, 80 cars and 12 car parks in this case. The random time value distribution functions $dist_{exec}$ and $dist_{open}$ that were used in this simulation were the same as the ones used in the simulation of experiments 1-4 and the generated set of events corresponded to a total of 2,971 transactions taking place over a period of 16.46 hours (i.e., 180.498 transactions per hour).

The results of this experiment are summarised in Table 6 and Figure 12. Table 6 shows the average and standard deviation of the decision delay (d-delay) as measured for every 2,000 events in the monitoring process, and the number of formula templates that had been created to represent different formula instances, and the average event waiting time at these points for this experiment. Figure 12 shows the average decision delay that was measured in this experiment along with results of Experiment 3 in which we had also monitored the formulas in *Set-2* against a set of events with smaller domain sizes.

#Events	**Average D-Delay (s)**		**St. Dev. Of D-Delay (s)**		**#Templates**		**Average Event Waiting Time (s)**	
	SD	LD	SD	LD	SD	LD	SD	LD
2000	0.102	0.105	0.052	0.054	263	281	0	0
4000	0.176	0.169	0.131	0.113	559	569	0	0
6000	0.250	0.249	0.200	0.203	851	839	0	0
8000	0.326	0.336	0.270	0.287	1148	1127	0	0
10000	0.404	0.411	0.348	0.357	1438	1401	0	0
12000	0.481	0.492	0.424	0.434	1714	1680	0	0
14000	0.556	0.570	0.490	0.495	2004	1983	0	0
16000	0.637	0.646	0.570	0.565	2279	2267	0	0
18000	0.716	0.722	0.646	0.639	2564	2553	0	0
20000	0.788	0.792	0.712	0.697	2837	2841	0	0

Table 6 – Monitoring Results for *Set-2* Using Larger Non Time Variable Domains (OI)

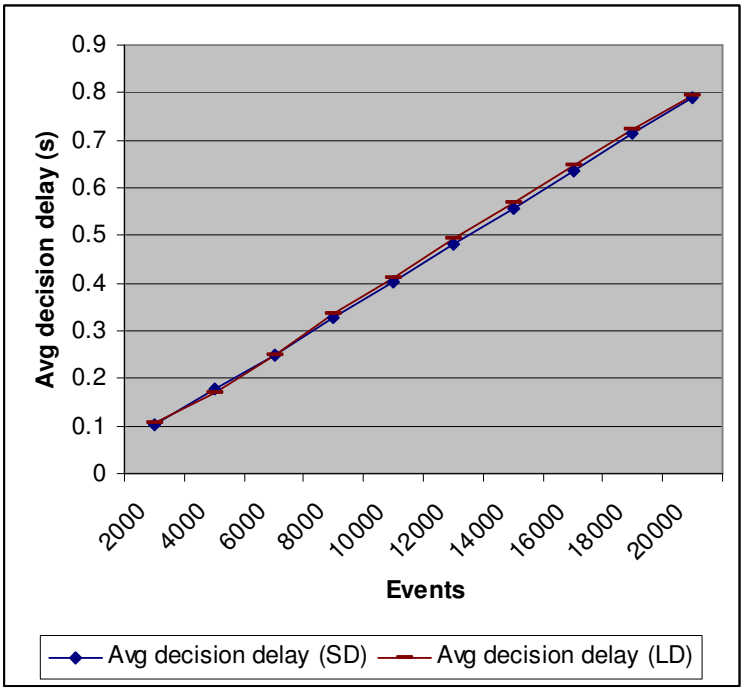


Figure 12 – Average decision delay in *Set-2*

As indicated by these results the increase in the size of the domains of the non time variables did not affect the performance of the dynamic validation tool. The absence of an effect of the size of the domains of the non time variable of the formulas that were monitored in our experiment was due to the structure of these formulas.

More specifically, the predicates in the formulas with constrained time variables had only non time variables which also appeared in the predicates with the unconstrained time variables. In rule R3 of Figure 4, for instance, the variable `carID` which is the only non time variable of the constrained predicate **`HoldsAt`**`(authorised_availability(carId),t1)` appears also in the unconstrained predicate of the formula **`Happens`**`(ic:makeAvailable(ID,status,sender,receiver,source,carId,loc),t1, R(t1,t1))`. In cases like this, when the unconstrained predicate in the formula is unified with the event, the variable binding that will result covers the variables of the constrained predicates as well and leaves no possibility for additional unification of the constrained predicates with subsequent events. Therefore, increments in the size of the domains of non time variables cannot lead to a proliferation in the number of the active templates of the formulas during the monitoring process. This phenomenon may be further explained through the example of the following monitoring rule[2]:

[2] The event structure in this rule does not include the sender, receiver, source and status variables of normal SERENITY events in order to simplify the discussion.

Happens(event(ID1, X), t1, R(t1,t1)) ∧

Happens(event(ID2, X,Y), t2, R(t1,t2) ⇒

Happens(event(ID3, Y), ct3, R(t2, t2+10)

In this example, when the predicate **Happens**(event(ID1, X), t1, R(t1,t1)) is unified with an event say, event(1,a,1) , the non time variable X of the predicate will take the value *a* and a template will be created to represent the partially unified formula. Subsequently, as the constrained predicate **Happens**(event(ID2, X,Y) in the rule has one more non time variable, namely Y, the partially instantiated template can be potentially unified with any events that have a value for X that is equal to *a* and any value for the variable Y. In this case, the maximum number of the complete instantiations of the partially instantiated template of the formula will depend on the size of the domain of Y. In the case of the main S&D properties that we have discussed in [7], however, we do not have any formulas with the same structure as the formula in the above example and, therefore, the size of the domain of the non-time variables of the formulas will not affect the performance of the monitor as the results of our 5th experiment that have been presented above confirmed.

3.2. Event Transmission Time

To investigate the time that it takes to transmit events from event captors to the dynamic validation prototype, we carried two experiments with varying event rates. For these experiments we developed a web service and a client for it. The client calls the web service at random time intervals. All the messages exchanged between the web service and the client are collected by the event captor which generates events to represent the messages and transmits them to the dynamic validation prototype (see [4] for more details). In these experiments, we measure the communication delay (t_d) using the following formula,

$$t_d = t_r - t_c$$

where t_c is the time point when a message is collected by the event captor and t_r is the time point when the corresponding event is received by the monitor manager. This formula was used as in the experiment both the event captor and the monitor manager were running on the same machine, and thus t_r and t_c were measures taken from the same machine clock.

In the first of the experiments, the client called the web service at time intervals that were generated randomly following the uniform distribution from 0 to 3 seconds. In the second experiment, the client called the web service at time intervals that are generated randomly following the uniform distribution from 0 to 5 seconds. In both experiments, the client to the web service was regulated to generated 10,000 events, and for each 1000 events we measured the average transmission delay and standard deviation. The results of these experiments are summarised below in Table 7 and Figure 13.

#Events	Experiment 6 (Event Transmission 1)		Experiment 7 (Event Transmission 2)	
	Average Event Transmission Delay (ms)	St. Dev	Average Event Transmission Delay (ms)	St. Dev
1000	23.057	52.323	22.806	33.664
2000	21.669	37.761	12.819	26.459
3000	15.013	32.394	12.934	22.709
4000	11.513	28.772	12.065	20.696
5000	9.422	26.139	9.838	19.127
6000	8.060	24.143	10.717	18.206
7000	7.145	22.558	10.469	17.458
8000	6.443	21.250	9.357	16.693
9000	5.885	20.155	8.450	16.061
10000	5.432	19.219	7.704	15.457

Table 7 – Average event transmission delay and standard deviation

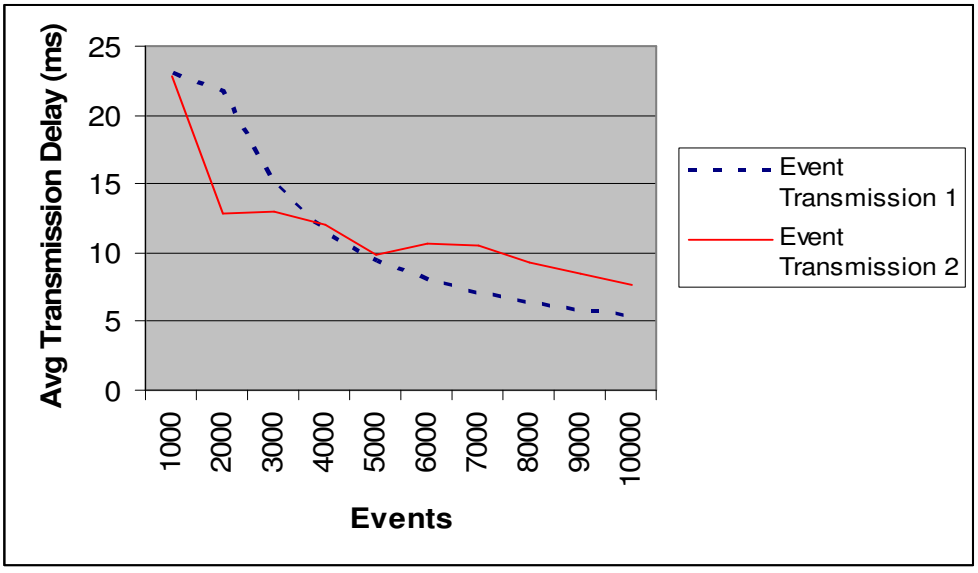


Figure 13 – Average event transmission delay

As it can be seen from Table 7 and Figure 13, the average event transmission delay was initially high (about 23 milliseconds) and then it felt steadily down to about 5-7 milliseconds. It should also be noted that the variability of the transmission delay also decreased steadily during the operation of the simple system used in our experiments (the initial standard deviation of the transmission delay for the first 1000 events in our experiments was 52.323 and 33.664 in the two experiments and went down by more than 50% to 19.219 and 15.457 at 10,000 events). Our explanation of this behaviour is that the initial relatively high delay was due to cost of establishing the initial socket connection which the captor used to communicate events to the dynamic validation prototype. Also it should be noted that the communication cost of events captured on the same machine where the dynamic validation tool runs is relatively low with respect to processing delays occurring during monitoring and, therefore, it is not expected to be a significant performance obstacle for the applicability of the dynamic validation prototype. Note, however, that the event transmission cost can be relatively high with respect to the cost of monitoring (delay in violation detection) if the captors and dynamic validation prototype are deployed on different machines. This scenario was not explored further in our experiment since the evaluated version of the dynamic validation tool does not support the processing of events whose timestamps have been generated by clocks other than the clock of the tool itself.

3.3. Effect of Event Capturing on Monitored Applications

In our final experiment (Experiment 8), we measured the effect of event capturing onto the response time of a monitored system implemented as a simple web service. In this experiment, we used a client which made 10,000 calls to the web service at random time intervals and measured the average and the standard deviation of the response time of the service when an event captor was deployed and when no event captor was deployed in every 1000 events. The event captor that was used in the experiment was SOAP message event captor that we had implemented based on the AXIS TCP Monitor utility that is described in [4].

Table 4 shows the average and standard deviation of the response time of the web service used in the experiment when an event captor was deployed (see *With Event Captor* column) and when no event captor was deployed (see *Without Event Captor* column). These statistics were measured for every 1000 calls in the experiment. Figure 14 shows a graph with the average response times in the same two cases.

As indicated by these results, the use of an event captor had a very significant effect on the response time of the service increasing it between 8.5 (in the range of 1-1,000 calls) and 19.9 times (in the range of 9001-10,000 calls). This effect appears to be very dramatic due to the fact that web-service deployed in the experiment had negligible processing time of its own. The type of the event captor that was deployed in the experiment was also partly responsible for the magnitude of the effect. More specifically, the SOAP message event captor that used is a general filtering mechanism that can be used to capture SOAP messages for all the services which are deployed in a server. To this end, extra processing is required to identify the service that a SOAP message is directed to and report it appropriately. Other types of event captors, such as workflow engine event captors, as the one that Mahbub and Spanoudakis have deployed in [6], have been reported to have a much less significant effect on the performance of the monitored system (an 18% increase in the response time of a workflow based system was reported in [6]). It should, however, be appreciated that the type of captor which was deployed in this experiment provides a general mechanism for applying different forms of controls onto the monitored system which other types of captors cannot support and,

therefore, the effect on the performance of the monitored system that we identified in this experiment might be necessary a necessary price to pay in cases where control is required.

Number of Calls	Without Event Captor		With Event Captor	
	Avg Response Time (ms)	Standard Deviation	Avg Response Time (ms)	Standard Deviation
1000	15.907	354.150	136.569	398.054
2000	10.073	250.602	123.646	282.233
3000	8.0263	204.714	115.005	230.908
4000	6.990	177.374	110.628	200.235
5000	6.332	158.721	108.001	179.2423
6000	5.935	144.963	105.975	163.729
7000	5.649	134.267	104.515	151.664
8000	5.447	125.651	103.463	141.950
9000	5.289	118.517	102.655	133.893
10000	5.134	112.483	102.072	127.110

Table 4 – Results of Experiment 8 – Average response time of web service

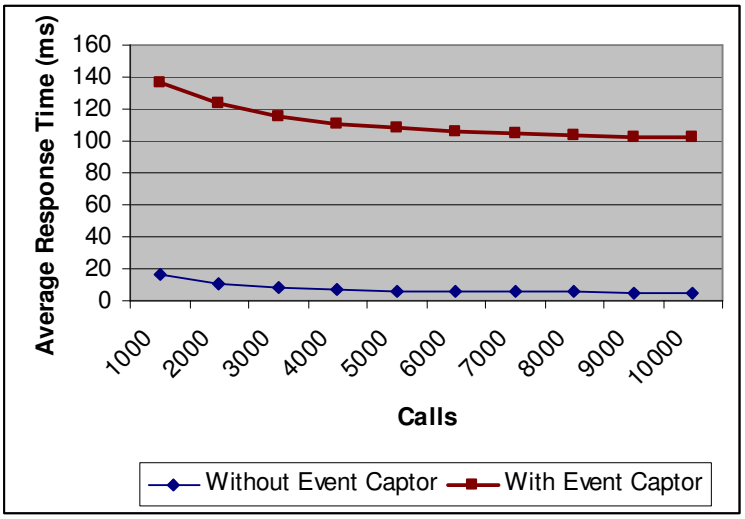


Figure 14 – Average response time with and without event captors in Experiment 8

4. Discussion and Possible Improvements

The results of the experiments, following the amendment of the dynamic validation prototype in order to prune templates during the monitoring process that we discussed in Section 3.1, demonstrated a very good average performance in the detection of violations of S&D properties. As indicated by the results of the experiments, the main factor that could have a significant adverse effect on the performance of the tool is the number of monitored formulas (this number appeared to affect the average decision delay exponentially).

To address this issue, we are investigating the possibility of creating a distributed version of the dynamic validation tool in which the *monitor manager* of the tool (see Figure 2 in [2]) will deploy different *monitors* distributed on separate machines and re-allocate monitoring formulas to them when the performance of a single monitor reaches a certain delay threshold. We are aware that in distributed monitoring, there will be additional costs arising from the communication between the distributed monitors and the monitor manager. Furthermore, the reallocation of monitoring formulas to distributed monitors will need also to be based on an analysis of dependencies between formulas as formulas which depend on each other will need to be allocated to a single monitor (a dependency between two formulas exist if the formulas share a common predicate as defined in [7]). The identification of such dependencies in a given set of monitoring formulas can be done offline and would need to be supported by a new tool integrated with the dynamic validation prototype.

To further improve the performance of the monitoring process we are also looking at the possibility of indexing templates over the predicates that they incorporate in order to make the process of searching through them in order to identify possible unifications with new events more efficient.

Also, we are looking into ways of minimising the fluent initiation and termination events (past time predicates) which are stored in the past predicate database of the dynamic validation tool. Deleting such events when they can no longer lead to the deduction of further information that is useful in the monitoring process can improve the performance of the monitor in certain cases (e.g. when it checks the validity of `HoldsAt` predicates as in rule R3 in our experiments).

Finally, we will need to investigate further the cost of transmitting events from event captors to monitors when these components are distributed to different machines. Apart from the normal communication costs which will arise in this case, we expect that there will be additional costs related to the need to order events which are captured by different captors operating with different clocks. This process will require additional processing that will increase the time between the capturing of events and the detection of violations from them. The ordering of different clock time stamps is a matter that the SERENITY team at City University is currently investigating.

5. Conclusions

In this deliverable, we have presented the results of an initial evaluation of V1 of the dynamic validation tool that was developed in SERENITY. The evaluation has focused on the performance of monitoring and event capturing and the effect of the latter on the systems which are being monitored.

More specifically, the main objective of our evaluation was to investigate the delay in the detection of a property violation by the dynamic validation tool and the ways in which this measure is affected by factors including the number and type of formulas which are being checked in a monitoring session (i.e. monitoring rules and/or assumptions) and the size of the domains of the variables of these formulas. The evaluation focused on the investigation of the average delay in detecting violations of S&D properties as earlier analysis has identified that the worst case computational complexity of the monitoring process is exponential.

Our evaluation was based on a series of simulations which generated random event sequences representing executions of a service based system that had been developed at City University. Using these event sequences we carried out experiments which showed that on average the detection of a violation is timely with the maximum average delay that was observed for typical S&D properties being in the order of seconds (the maximum observed average delay was 3.8 seconds). The experiments also indicated that the number of formulas which are being monitored affects performance but that the size of the domains of the variables did not have an effect. This was the case when the formulas used in monitoring had a form where the constrained predicates did not have any non time variables other than those appearing in the unconstrained predicates. The carried out experiments also showed that the incorporation of assumptions in the monitoring process did not have a significant effect on performance.

Finally, the experiments have indicated that the communication costs of transmitting events from event captors to the dynamic validation tool when both these components are running on the same machine are low and that event capturing has a significant effect on the performance of the system that is being monitored. The latter observation relates to captors which act as filters to the process that is being monitored and therefore are capable of applying certain forms of control to them.

The different rule and event sets that we used in the experiments which have been presented in this report are available in the file `A4.D3.2_RulesAndEvents.zip` that is available from:

https://bscw.sit.fraunhofer.de/bscw/bscw.cgi/911075

As part of the initial evaluation that we carried out we introduced certain optimisations in the monitoring engine of the dynamic validation tool. These optimisations will be released with the second version of the tool. Further optimisations are also under investigation for incorporation in the next version of the tool including the implementation of index-based searches of monitoring formula templates during the monitoring process and the development of a distributed version of the dynamic validation tool.

Appendix A. Specification of the Car Rental System (CRS)

A.1. BPEL Specification of the CRS

```
<process name="carServiceProcess"
     targetNamespace="http://carservice.org/carserviceprocessing"
     suppressJoinFailure="yes"
     xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
     xmlns:sns="http://carservice.org/wsdl/OnlineRenter"
     xmlns:crns="http://tempuri.org/services/CarReg"
     xmlns:csns="http://tempuri.org/services/CustomerReg">

   <variables>
     <variable name="authRes"
                      messageType="csns:authenticateResponse"/>
     <variable name="authReq"
                       messageType="csns:authenticateRequest"/>
     <variable name="isAvailReq"
                       messageType="crns:isAvailableRequest"/>
     <variable name="isAvailRes"
                        messageType="crns:isAvailableResponse"/>
     <variable name="depReq" messageType="sns:departRequest"/>
     <variable name="depRes" messageType="sns:departResponse"/>
     <variable name="entReq" messageType="sns:enterRequest"/>
     <variable name="entRes" messageType="sns:enterResponse"/>
     <variable name="retKeyReq" messageType="sns:retKeyRequest"/>
     <variable name="retKeyRes" messageType="sns:retKeyResponse"/>
     <variable name="mkUnAvailReq"
                       messageType="crns:makeUnAvailableRequest"/>
     <variable name="mkUnAvailRes"
                      messageType="crns:makeUnAvailableResponse"/>
     <variable name="mkAvailReq"
                        messageType="crns:makeAvailableRequest"/>
     <variable name="mkAvailRes"
                       messageType="crns:makeAvailableResponse"/>
     <variable name="bpelReq" messageType="sns:request"/>
     <variable name="bpelRes" messageType="sns:response"/>
   </variables>

   <partners>
```

```
    <partner name="CRS" serviceLinkType="sns:CarRenterLT"
                                        myRole="RentManager"/>
    <partner name="CMS" serviceLinkType="sns:CustomerManagerLT"
                              partnerRole="CustomerManager"/>
    <partner name="CRMS" serviceLinkType="sns:CarManagerLT"
                                      partnerRole="CarManager"/>
</partners>

<correlationSets>
     <correlationSet name="locInfo" properties="sns:locs"/>
      <correlationSet name="carInfo" properties="sns:car_id"/>
      <correlationSet name="custInfo" properties="sns:cust_id"/>
      <correlationSet name="locAndCarInfo" properties="sns:locs
                                                 sns:car_id"/>
</correlationSets>

<flow>
    <links>
          <link name="receive-to-auth"/>
          <link name="auth-to-check"/>
          <link name="check-to-car"/>
          <link name="car-to-reply"/>
          <link name="check-to-noCar"/>
          <link name="noCar-to-reply"/>
          <link name="auth-to-no"/>
          <link name="no-to-reply"/>
         <link name="enter-to-retKey"/>
          <link name="release-to-depNotDep"/>
    </links>

    <receive name="receive1" partner="CRS" portType="sns:CarRenter"
         operation="receiveRequest" variable="bpelReq"
                                          createInstance="yes">
          <source name="rToa" linkName="receive-to-auth"/>
          <correlations>
                <correlation set="locInfo" initiate="yes"/>
                <correlation set="custInfo" initiate="yes"/>
                <correlation set="locAndCarInfo" initiate="yes"/>
          </correlations>
    </receive>
    <assign name="assign1">
```

```
        <target linkName="auth-to-no"/>
        <source linkName="no-to-reply"/>
        <copy>
              <from expression="'Customer Not Authenticated'"/>
              <to variable="bpelRes" part="carId"/>
        </copy>
</assign>

<sequence>
        <target linkName="auth-to-check"/>
        <assign name="assign2">
              <copy>
                    <from variable="bpelReq" part="loc"/>
                    <to variable="isAvailReq" part="loc"/>
              </copy>
        </assign>
        <invoke name="invokeIsAvail" partner="CRMS"
              portType="crns:CarReg" operation="isAvailable"
              inputVariable="isAvailReq"
              outputVariable="isAvailRes">
               <source linkName="check-to-noCar"
                    transitionCondition="bpws:getVariableData
                       ('isAvailRes','carId') = 'null'"/>
              <source linkName="check-to-car"
                    transitionCondition="bpws:getVariableData
                      ('isAvailRes' , 'carId') != 'null'"/>

              <correlations>
                   <correlation set="carInfo" initiate="yes"
                                                pattern="in"/>
                     <correlation set="locAndCarInfo"
                                 initiate="yes" pattern="in"/>
              </correlations>
        </invoke>
</sequence>

<assign name="assign3">
        <target linkName="check-to-car"/>
        <source linkName="car-to-reply"/>
        <copy>
              <from variable="isAvailRes" part="carId"/>
```

```
                <to variable="bpelRes" part="carId"/>
        </copy>
</assign>
<assign name="assign4">
        <target linkName="check-to-noCar"/>
        <source linkName="noCar-to-reply"/>
        <copy>
                <from expression="'Car Not Available'"/>
                <to variable="bpelRes" part="carId"/>
        </copy>
</assign>

<sequence>
        <target linkName="receive-to-auth"/>
        <assign name="assign5">
                <copy>
                        <from variable="bpelReq" part="custId"/>
                        <to variable="authReq" part="custId"/>
                </copy>
        </assign>
        <invoke name="invokeAuth" partner="CMS"
            portType="csns:CustomerReg" operation="authenticate"
            inputVariable="authReq" outputVariable="authRes">

            <source name="aToc" linkName="auth-to-check"
               transitionCondition="(bpws:getVariableData
               ('authRes' , 'authenticateReturn') = true())"/>
               <source name="aTon" linkName="auth-to-no"
                transitionCondition="bpws:getVariableData
                ('authRes' , 'authenticateReturn') = false()"/>
        </invoke>
</sequence>

<reply name="reply" partner="CRS" portType="sns:CarRenter"
                operation="receiveRequest" variable="bpelRes">
        <target linkName="car-to-reply"/>
      <source linkName="release-to-depNotDep"/>
        <correlations>
                <correlation set="carInfo"/>
        </correlations>
</reply>
```

```
<reply name="reply" partner="CRS" portType="sns:CarRenter"
            operation="receiveRequest" variable="bpelRes">
      <target linkName="noCar-to-reply"/>
      <target linkName="no-to-reply"/>
      <correlations>
      <correlation set="carInfo"/>
      </correlations>
</reply>

<sequence>
   <receive name="receive2" partner="CRS"
        portType="sns:CarRenter" operation="enter"
                                          variable="entReq">
      <correlations>
           <correlation set="locInfo" initiate="yes"/>
           <correlation set="carInfo" initiate="no"/>
      </correlations>
      </receive>
     <reply partner="CRS" portType="sns:CarRenter"
                operation="enter" variable="entRes"/>
       <source linkName="enter-to-retKey"/>
</sequence>
<pick>
  <target linkName="enter-to-retKey"/>
  <onMessage partner="CRS" portType="sns:CarRenter"
              operation="returnKey" variable="retKeyReq">
    <correlations>
      <correlation set="locInfo"/>
           <correlation set="carInfo"/>
    </correlations>
   <sequence>
        <assign name="assign6">
            <copy>
                 <from variable="retKeyReq" part="carId"/>
                 <to variable="mkAvailReq" part="carId"/>
            </copy>
         </assign>
         <assign name="assign7">
             <copy>
                  <from variable="retKeyReq" part="loc"/>
```

```
                    <to variable="mkAvailReq" part="loc"/>
                </copy>
             </assign>
             <invoke name="mkAvail1" partner="CRMS"
                portType="crns:CarReg" operation="makeAvailable"
                inputVariable="mkAvailReq"
                                    outputVariable="mkAvailRes">

                <correlations>
                      <correlation set="locInfo" pattern="out"/>
                      <correlation set="carInfo" pattern="out"/>
                </correlations>
             </invoke>
             <reply partner="CRS" portType="sns:CarRenter"
                    operation="returnKey" variable="retKeyRes"/>
       </sequence>
    </onMessage>
    <onAlarm for="'PT90S'">
            <empty/>
    </onAlarm>
</pick>
<pick>
    <target linkName="release-to-depNotDep"/>
    <onMessage partner="CRS" portType="sns:CarRenter"
                    operation="depart" variable="depReq">
        <correlations>
         <correlation set="locAndCarInfo" initiate="no"/>
        </correlations>
       <sequence>
          <assign name="assign8">
             <copy>
                  <from variable="depReq" part="carId"/>
                  <to variable="mkUnAvailReq" part="carId"/>
             </copy>
           </assign>
          <assign name="assign9">
              <copy>
                 <from variable="depReq" part="loc"/>
                 <to variable="mkUnAvailReq" part="loc"/>
              </copy>
           </assign>
          <assign name="assign10">
```

```
            <copy>
               <from variable="bpelReq" part="custId"/>
               <to variable="mkUnAvailReq" part="custId"/>
            </copy>
         </assign>
        <invoke name="mkUnAvail" partner="CRMS"
            portType="crns:CarReg" operation="makeUnAvailable"
            inputVariable="mkUnAvailReq"
                        outputVariable="mkUnAvailRes">

            <correlations>
            <correlation set="locInfo" pattern="out"/>
                  <correlation set="carInfo" pattern="out"/>
            <correlation set="custInfo" pattern="out"/>
            </correlations>
        </invoke>
        <reply partner="CRS" portType="sns:CarRenter"
                     operation="depart" variable="depRes"/>
    </sequence>
</onMessage>
<onAlarm for="'PT30S'">
    <sequence>
       <assign name="assign11">
            <copy>
               <from variable="isAvailRes" part="carId"/>
               <to variable="mkAvailReq" part="carId"/>
            </copy>
         </assign>
       <assign name="assign12">
            <copy>
               <from variable="bpelReq" part="loc"/>
               <to variable="mkAvailReq" part="loc"/>
            </copy>
         </assign>
       <invoke name="mkAvail2" partner="CRMS"
          portType="crns:CarReg" operation="makeAvailable"
          inputVariable="mkAvailReq"
                        outputVariable="mkAvailRes">

            <correlations>
               <correlation set="locInfo" pattern="out"/>
               <correlation set="carInfo" pattern="out"/>
```

```
                    </correlations>
                  </invoke>
              </sequence>
              </onAlarm>
            </pick>
  </flow>
</process>
```

A.2. WSDL Specification of the CRS

```
<definitions
   targetNamespace="http://carservice.org/wsdl/OnlineRenter"
   xmlns:tns="http://carservice.org/wsdl/OnlineRenter"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:slnk="http://schemas.xmlsoap.org/ws/2003/03/service-link/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:crns="http://tempuri.org/services/CarReg"
   xmlns:csns="http://tempuri.org/services/CustomerReg">


  <import namespace="http://tempuri.org/services/CustomerReg"
      location=
      "http://138.40.91.72:8080/wstk/CustomerReg/CustomerReg.wsdl"/>

  <import namespace="http://tempuri.org/services/CarReg"
     location="http://138.40.91.72:8080/wstk/CarReg/CarReg.wsdl"/>


  <message name="request">
      <part name="custId" type="xsd:string"/>
      <part name="loc" type="xsd:string"/>
  </message>

  <message name="response">
      <part name="carId" type="xsd:string"/>
  </message>

  <message name="departRequest">
      <part name="carId" type="xsd:string"/>
      <part name="loc" type="xsd:string"/>
  </message>
```

```
<message name="departResponse">

</message>

<message name="enterRequest">
    <part name="carId" type="xsd:string"/>
    <part name="loc" type="xsd:string"/>
</message>

<message name="enterResponse">

</message>

<message name="retKeyRequest">
    <part name="carId" type="xsd:string"/>
    <part name="loc" type="xsd:string"/>
</message>

<message name="retKeyResponse">

</message>

<portType name="CarRenter">
     <operation name="receiveRequest">
           <input message="tns:request"/>
           <output message="tns:response"/>
     </operation>
    <operation name="depart">
          <input message="tns:departRequest"/>
          <output message="tns:departResponse"/>
     </operation>
    <operation name="enter">
          <input message="tns:enterRequest"/>
          <output message="tns:enterResponse"/>
     </operation>
    <operation name="returnKey">
          <input message="tns:retKeyRequest"/>
          <output message="tns:retKeyResponse"/>
     </operation>
</portType>
```

```
<slnk:serviceLinkType name="CarRenterLT">
     <slnk:role name="RentManager">
           <portType name="tns:CarRenter"/>
     </slnk:role>
</slnk:serviceLinkType>

<slnk:serviceLinkType name="CustomerManagerLT">
  <slnk:role name="CustomerManager">
    <portType name="csns:CustomerReg"/>
  </slnk:role>
</slnk:serviceLinkType>

<slnk:serviceLinkType name="CarManagerLT">
  <slnk:role name="CarManager">
    <portType name="crns:CarReg"/>
  </slnk:role>
</slnk:serviceLinkType>

 <property name="car_id" type="xsd:string"/>
 <property name="locs" type="xsd:string"/>
 <property name="cust_id" type="xsd:string"/>

<propertyAlias propertyName="tns:cust_id"
                     messageType="tns:request"
                     part="custId"
                     query="/custId"/>

<propertyAlias propertyName="tns:cust_id"
                     messageType="csns:authenticateRequest"
                     part="custId"
                     query="/custId"/>

<propertyAlias propertyName="tns:locs"
                     messageType="tns:departRequest"
                     part="loc"
                     query="/loc"/>

 <propertyAlias propertyName="tns:locs"
                     messageType="tns:request"
                     part="loc"
```

```
                        query="/loc"/>

    <propertyAlias propertyName="tns:locs"
                        messageType="crns:isAvailableRequest"
                        part="loc"
                        query="/loc"/>

   <propertyAlias propertyName="tns:car_id"
                        messageType="crns:isAvailableResponse"
                        part="isAvailableReturn"
                        query="/isAvailableReturn"/>

   <propertyAlias propertyName="tns:car_id"
                        messageType="tns:departRequest"
                        part="carId"
                        query="/carId"/>

   <!-- The service name and the TNS represent my service ID QName -->
   <service name="carServiceBP"/>

</definitions>
```

A.3. WSDL Specification of the Car Information System (IS)

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
      targetNamespace="http://tempuri.org/services/CarReg"
      xmlns="http://schemas.xmlsoap.org/wsdl/"
      xmlns:apachesoap="http://xml.apache.org/xml-soap"
       xmlns:impl=http://tempuri.org/services/CarReg  xmlns:intf="http://tempuri.org/services/CarReg"
       xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
       xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
       xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema">


   <wsdl:message name="makeAvailableResponse">


   </wsdl:message>


   <wsdl:message name="makeUnAvailableRequest">
      <wsdl:part name="carId" type="xsd:string"/>
      <wsdl:part name="custId" type="xsd:string"/>
      <wsdl:part name="loc" type="xsd:string"/>
```

```
</wsdl:message>

<wsdl:message name="isAvailableResponse">
   <wsdl:part name="carId" type="xsd:string"/>
</wsdl:message>

<wsdl:message name="makeAvailableRequest">
   <wsdl:part name="carId" type="xsd:string"/>
   <wsdl:part name="loc" type="xsd:string"/>
</wsdl:message>

<wsdl:message name="isAvailableRequest">
   <wsdl:part name="loc" type="xsd:string"/>
</wsdl:message>

<wsdl:message name="makeUnAvailableResponse">

</wsdl:message>

<wsdl:portType name="CarReg">
   <wsdl:operation name="isAvailable" parameterOrder="loc">
      <wsdl:input message="impl:isAvailableRequest"
                              name="isAvailableRequest"/>
      <wsdl:output message="impl:isAvailableResponse"
                             name="isAvailableResponse"/>

   </wsdl:operation>

   <wsdl:operation name="makeUnAvailable" parameterOrder="carId
                                                    custId loc">

      <wsdl:input message="impl:makeUnAvailableRequest"
                                  name="makeUnAvailableRequest"/>
      <wsdl:output message="impl:makeUnAvailableResponse"
                                  name="makeUnAvailableResponse"/>
   </wsdl:operation>

   <wsdl:operation name="makeAvailable" parameterOrder="carId
                                                          loc">
      <wsdl:input message="impl:makeAvailableRequest"
                                    name="makeAvailableRequest"/>
```

```
      <wsdl:output message="impl:makeAvailableResponse"
                                      name="makeAvailableResponse"/>
   </wsdl:operation>

</wsdl:portType>

<wsdl:binding name="CarRegServiceSoapBinding" type="impl:CarReg">
   <wsdlsoap:binding style="rpc"
               transport="http://schemas.xmlsoap.org/soap/http"/>
   <wsdl:operation name="isAvailable">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="isAvailableRequest">
         <wsdlsoap:body
         encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
         namespace="http://tempuri.org/services/CarReg"
                                                 use="encoded"/>
      </wsdl:input>
      <wsdl:output name="isAvailableResponse">
         <wsdlsoap:body
         encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="http://tempuri.org/services/CarReg"
                                                  use="encoded"/>
      </wsdl:output>
   </wsdl:operation>

   <wsdl:operation name="makeUnAvailable">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="makeUnAvailableRequest">
         <wsdlsoap:body
         encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="http://tempuri.org/services/CarReg"
                                                  use="encoded"/>
      </wsdl:input>
      <wsdl:output name="makeUnAvailableResponse">
         <wsdlsoap:body
         encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="http://tempuri.org/services/CarReg"
                                                 use="encoded"/>
      </wsdl:output>
   </wsdl:operation>
```

```
      <wsdl:operation name="makeAvailable">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="makeAvailableRequest">
            <wsdlsoap:body
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
             namespace="http://tempuri.org/services/CarReg"
                                                    use="encoded"/>
         </wsdl:input>

         <wsdl:output name="makeAvailableResponse">
            <wsdlsoap:body
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
             namespace="http://tempuri.org/services/CarReg"
                                                    use="encoded"/>
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>

   <wsdl:service name="CarRegService">
      <wsdl:port binding="impl:CarRegServiceSoapBinding"
                                          name="CarRegService">
         <wsdlsoap:address location=
             "http://138.40.91.72:8080/wstk/services/CarRegService"/>
      </wsdl:port>
   </wsdl:service>
</wsdl:definitions>
```

References

[1] Andrews T. et al. (2003), Business Process Execution Language for Web Services, v1.1, http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf (as last seen on 15/6/07)

[2] Androutsopoulos K., Ballas C., Kloukinas C., Mahbub K., Spanoudakis G (2007), V1 of Dynamic Validation Prototype, SERENITY Deliverable A4.D3.1, February 2007.

[3] Bonato R., et al., (2007), Evaluation Criteria, SERENITY Deliverable A47.D5.1, February 2007.

[4] Kloukinas C., Ballas C., Presenza D., Spanoudakis G. (2007), Basic Set of Information Collection Mechanisms for S&D Monitoring, SERENITY Deliverable A4.D2.2, October 2006.

[5] Mahbub K. (2007), Requirements Monitoring of Service Based Systems, PhD Dissertation, Department of Computing, City University.

[6] Mahbub K. Spanoudakis G. (2007), Monitoring WS Agreements: An Event Calculus Based Approach, In *Test and Analysis of Service Oriented Systems*, (eds) L. Baresi, E. diNitto, Springer Verlang (to appear)

[7] Spanoudakis G., Kloukinas C., Androutsopoulos K. (2007), Towards Security Monitoring Patterns, Proceedings of the 22nd Annual ACM Symposium on Applied Computing, Technical Track on Software Verification, March 2007

[8] Spanoudakis G., Mahbub K. (2006), Non Intrusive Monitoring of Service Based Systems, International Journal of Cooperative Information Systems, Vol. 15, No. 3 (2006), 325-358