Software Engineering Group
Department of Computer Science
Nanjing University
http://seg.nju.edu.cn

Technical Report No. NJU-SEG-2013-IJ-004

2013-IJ-004

WS-PSC Monitor: A tool chain for monitoring BPEL-based web service composition with scenario-based specifications

Pengcheng Zhang, Hareton Leung, Wenrui Li, Xuandong Li

Runtime Verification Volume 6418 2010

Most of the papers available from this document appear in print, and the corresponding copyright is held by the publisher. While the papers can be used for personal use, redistribution or reprinting for commercial purposes is prohibited.

www.ietdl.org

Published in IET Software
Received on 1st June 2011
Revised on 19th December 2012
Accepted on 27th February 2013
doi: 10.1049/iet-sen.2012.0047

ISSN 1751-8806

Web services property sequence chart monitor: a tool chain for monitoring BPEL-based web service composition with scenario-based specifications

Pengcheng Zhang[1,2], *Hareton Leung*[3], *Wenrui Li*[4], *Xuandong Li*[1]

[1]*State Key Laboratory of Novel Software Technology, Nanjing University, Nanjing 210093, People's Republic of China*
[2]*College of Computer and Information Engineering, Hohai University, Nanjing 210098, People's Republic of China*
[3]*Department of Computing, Hong Kong Polytechnic University, Hong Kong 999077, People's Republic of China*
[4]*School of Mathematics and Information Technology, Nanjing Xiaozhuang University, Nanjing 211171, People's Republic of China*
E-mail: pchzhang@seg.nju.edu.cn; Lxd@nju.edu.cn

Abstract: Web service composition is a new paradigm to develop distributed and reactive software-intensive systems. Owing to the autonomous nature of basic services, the validation of composite service must be extended from design-time to run-time. Here, the authors describe a novel tool chain called web services property sequence chart monitor to monitor temporal, timing and probabilistic properties in composite service based on scenario-based property specifications called property sequence chart, timed property sequence chart and probabilistic timed property sequence chart, respectively. The tool chain provides a completely graphical front-end that eliminates the need to deal with any particular textual and logical formalism. Furthermore, the framework and implementation detail of the tool chain are also presented. Finally, the feasibility and usability of the tool have been validated by the case studies and performance measurement.

For Research Only

1 Introduction

In recent years, the idea of software as a service has added a new paradigm to the service-oriented architecture (SOA). In SOA, basic services are seen as autonomous agents acting according to certain contracts. For example, through workflow languages such as BPEL [1], service requestors may compose existing basic services to provide more powerful composite services. For such systems, verification is particularly challenging as the overall behaviour depends heavily on the participating agents, which renders the analysis of such systems prior to execution almost impossible [2]. Consequently, run-time analysis techniques, such as run-time monitoring that detects the behaviours of system against the desired properties [3], are being pursued as a lightweight verification technique complementing traditional verification techniques at design-time, such as model checking and testing.

The specifications for monitored properties focus mostly on logic-based or scenario-based specification formalisms [4]. Logic-based formalisms are often more expressive than scenario-based formalisms. However, the logic-based formalisms have the following limitations:

- *Lack of intuitiveness:* Properties that are simply captured and described in intuitive way by the natural languages are not easily specified in logic-based specifications.
- *State-based:* The atomic propositions of a logic-based specification can only be state propositions, where messages (events) cannot be easily expressed. Consequently, it is not straightforward to use these logic-based specifications to specify the properties of event-based web service compositions.
- *Trace error interpretation:* In conventional logic-based run-time monitoring, even if an error trace is detected, it is difficult for the developers to debug the trace and pinpoint the problem due to the internal complexity of logic-based specifications.

The first and third limitations have also been emphasised by Holzmann [5], who recognised that the 'most underestimated problem in applications of automated tools to software verification is the problem of accurately capturing the correctness requirements that have to be verified', and Dwyer *et al.* [6] who recognised the difficulty in writing properties correctly.

The second limitation implies great challenge in the characterisation of event synchronisation or causal relation. For example, an informal requirement can be described as: 'if service A sends message m_1 to service B, and service B sends m_2 to service C (in any order), then C must send m_3 to service A within three time units'. Using traditional logic-based specification to specify these event-based interactions is very hard.

© The Institution of Engineering and Technology 2013

Scenario-based formalisms, such as message sequence chart (MSC) [7] and UML 2.0 sequence diagram (UML 2.0 SD) [8], provide a graphical modelling formalism that is widely accepted in industrial practice. Furthermore, all these formalisms provide event-based notations and, consequently, are suitable to specify event-based web service composition properties easily, intuitively and simply. However, the expressiveness of such kinds of graphical specifications is rather weak due to their ambiguous semantics.

To deal with these problems, Damm and Harel [9] propose live sequence chart (LSC) as a visual formalism for scenario-based requirement specification. LSC makes essential extensions to MSC by adding modalities. The existential and universal modalities represent the provisional and mandatory requirements, respectively. The power of LSC lies in that a universal LSC chart can optionally contain a pre-chart, which specifies the scenario which, if successfully executed (or matched), forces the system to satisfy the scenario given in the actual chart body (i.e. the main chart). Furthermore, the LSC language is unambiguous because its semantics is strictly defined [9]. However, the LSC formalism still has the following limitations:

- *Complexity:* Compared with traditional MSC and UML 2.0 SD, LSC is still more difficult to use due to complex concepts such as 'cut' and 'location'. The problem is that we need to devise a new property specification formalism that can overcome the limitations of MSC and SD, and is as close to them as possible.
- *Lack of ability to specify chain properties:* Another main disadvantage of LSC is that it cannot clearly support specifying chain constraints. In fact a chain constraint allows the specification of what can be performed before and after exchanging a message. Chains are very important to specify causes, effects, precedence and response relations.
- *Lack of timed and probabilistic extensions:* Currently, there is only one time version of LSC, called time-enriched LSC. However, the semantics of time-enriched LSC is not clearly defined. Furthermore, to the best of our knowledge, there is no probabilistic version of LSC to specify probabilistic properties, which are also very important for certain software systems.

Consequently, to deal with these limitations, this paper focuses on monitoring BPEL-based web service composition based on newly proposed scenario-based specification formalisms called property sequence chart (PSC) [10], timed property sequence chart (TPSC) [11] and probabilistic time property sequence chart (PTPSC) [12]. We envisage PSC/TPSC/PTPSC as a nice complement to the existing property specification languages for monitoring web service composition, with the following desirable features:

- *Intuitiveness:* PSC/TPSC/PTPSC specifications have the necessary language constructs to describe a variety of causality and chain properties. Furthermore, as extensions of a subset of UML 2.0 SD, they are as close to UML 2.0 SDs as possible and are more intuitive in capturing scenario-based user requirements than the logic-based specifications. Finally, PSC/TPSC/PTPSC can provide a completely graphical front-end that eliminates the need to deal with any particular textual formalism.
- *Expressiveness:* PSC/TPSC/PTPSC can be used to specify temporal, timing and probabilistic properties of BPEL-based web service composition, respectively. Furthermore, the expressiveness of PSC/TPSC/PTPSC has already been validated by property specification patterns (PSPs) [6], real-time specification patterns [13] and probabilistic specification patterns [14].
- *Trace error display:* PSC/TPSC/PTPSC can overcome the trace error interpretation problems by providing the possibility of tracing the error back to the scenario-based requirement specifications. Thus, it is easy for the developers to debug and find the trace errors related to both the system models and the requirements.

In summary, this paper makes the following major contributions: We capture a web service composition scenario to be monitored using PSC/TPSC/PTPSC specifications. For PSC specifications, we first obtain a behaviour-equivalent finite automaton (FA) from this chart according to the formal semantics, then we check whether the run-time traces are the accepting language of the generated FA. For TPSC specifications, we first obtain a behaviour-equivalent timed automaton (TA) from this chart according to the formal semantics, then we check whether the run-time traces are the accepting language of the generated TA. For PTPSC specifications, a probabilistic monitor combining TA and a sequential probability ratio test (SPRT) procedure is automatically generated to check whether the run-time traces satisfy the probabilistic properties. A corresponding tool chain called WS-PSC monitor is developed. To demonstrate the generality of our proposed approach and measure its performance, some evaluations of WS-PSC monitor are performed.

The rest of the paper is organised as follows: Section 2 presents a running example to be used in the paper. This example illustrates temporal, timing and probabilistic properties in web service composition. Section 3 introduces the basic concepts of PSC, TPSC and PTPSC specifications. Section 4 details the framework and implementation of WS-PSC tool. The case studies and performance evaluations of the tool are presented in Section 5. Section 6 compares our approach to related work. Section 7 concludes the paper and gives a list of future work.

2 Running example

We use an online medical assistant (OMA) composite service as a running example in this paper. OMA is a system proposed by IBM China [http://www-31.ibm.com/smb/cn/industries/healthcare/] and is becoming popular in China. It provides patients, doctors and managers in medical institutes with useful medical services. OMA includes seven services:

- Medical service agent (MSA) is a medium between patients and medical institutes, which makes an appointment and referral to specialty hospitals.
- Primary care hospital (PCH) is a hospital that offers primary care services to patients, including routine vaccinations and basic consultation.
- Medical insurance centre (MIC) can verify the patient's medicard that provides patient financing for medical treatments.
- High health sector (HHS) can provide advanced care to patients who cannot be treated in a primary care facility alone.

For Research Only

© The Institution of Engineering and Technology 2013

• Medical management institute (MMI) can store simple registration information of all the medical institutes.
• Electronic medical record (EMR) is a computerised medical record created in an organisation that delivers care, such as a hospital and doctor's office. EMR tends to be a part of a local stand-alone health information system that allows storage, retrieval and modification of records.
• Drug logistics company (DLC) can receive a drug list for the patient from MSA, then delivers the medicine to the patient.

Fig. 1 describes the typical scenarios between the MSA and its partners using the BPEL designer. A patient, who wants to see a doctor, sends query (SubmitQuery) to MSA. The MSA sends patient condition (SendPatientCondition) to the MMI, and returns several medical institutes that meet the patient's requirements. Then, the patient chooses a hospital (ChoosePrimaryHospital) and makes an appointment (MakeAppointment) with a PCH through the MSA, and then goes to this hospital. The doctor in PCH has the patient's basic health records and personal information queried from EMR. The doctor can directly diagnose the patient and return the results (DiagnosisResult). Any authorised doctor in PCH can browse the patient's previous treatment records through EMR, so that the patient can get the same care anywhere. Then, the diagnostic records will be uploaded to EMR system (UpdateEMR) by the doctor, and diagnosis is finished (FinishDiagnosis). The detailed cost (SendDetailCost) and corresponding medicine (DeliveryLogisticsCompany) are then sent to the patient. The PCH also needs to record the patient's diagnosis results (DetailDiagonsisResult) for future visits. If the doctor in PCH cannot diagnose the disease, then the MSA calls the high-level health sectors (HHS) (CallHighHealthSectors). The doctors in PCH and specialists in HHS can conduct remote consultations (RemoteVideoConsultations) to establish the diagnostic and treatment regimen via video conferences. In this process, the patient waits for the further diagnosis result (FurtherDiagnosisResult). According to the result, if the doctor in PCH can do the treatment, the process will be finished like the normal case. If a change of hospital is needed (ChangeHospital), the patient is transferred into the HHS (HighHealthSector) and the process is closed. If medical researchers or specialists are interested in some special medical cases, they can also query (Query) EMR to obtain the detailed records (SendPC) for pathological and statistical analysis.

Three temporal properties of the process can be specified as follows:

• *Prop1:* After the patient submits a request to the MSA, MSA asks MMI to query a medical institute that meets the patient's requirements. Then MSA sends a hospital list returned by MMI to the patient who selects one hospital to make an appointment.
• *Prop2:* If diagnosis is finished, PCH will ask MSA to record the diagnostic report into the EMR and transmit the result to the DLC. Furthermore, this information is also transmitted to MIC for promoting reimbursement process. The PCH will also receive the patient's diagnosis.
• *Prop3:* A personal health record in EMR cannot be released without the patient's permission.

These temporal properties should be monitored at run-time.

Time plays an important role for time-critical composite services. Thus, some timing properties need to be monitored at run-time. If these properties are not satisfied, some serious consequences may happen. In our example, if a patient's request cannot be replied within the desired time, the patient's diagnosis will be delayed, which may have serious consequences. Two timing constraints can be specified as follows:

• *Prop4:* After the patient submits a request to the MSA, within 30 s, MSA will send the hospital list returned by MMI to the patient.
• *Prop5:* After completing the diagnosis, within 30 min, PCH will ask MSA to record the diagnostic report into the EMR, and transmit the result to the essential DLC, MIC and the PCH.

Sometimes, it is very difficult to assure the strict correctness of timing properties.

Recently, there is an increasing research on monitoring probabilistic properties in composite service, since they can be used to formulate reliability, availability, safety and performance requirements. Consequently, two probabilistic properties of the process can be specified as follows:

• *Prop6:* After the patient submits a request to the MSA, within 30 s and with 90% probability, MSA will send the hospital list returned by MMI to the patient.
• *Prop7:* After completing the diagnosis, there is 95% probability that within 30 min PCH will ask MSA to record the diagnostic report into the EMR, transmit the result to the essential DLC, MIC and the PCH.

3 Formal operational semantics of PSC/TPSC/PTPSC

This section will give formal operational semantics of PSC, TPSC and PTPSC for run-time monitoring purpose according to [10–12].

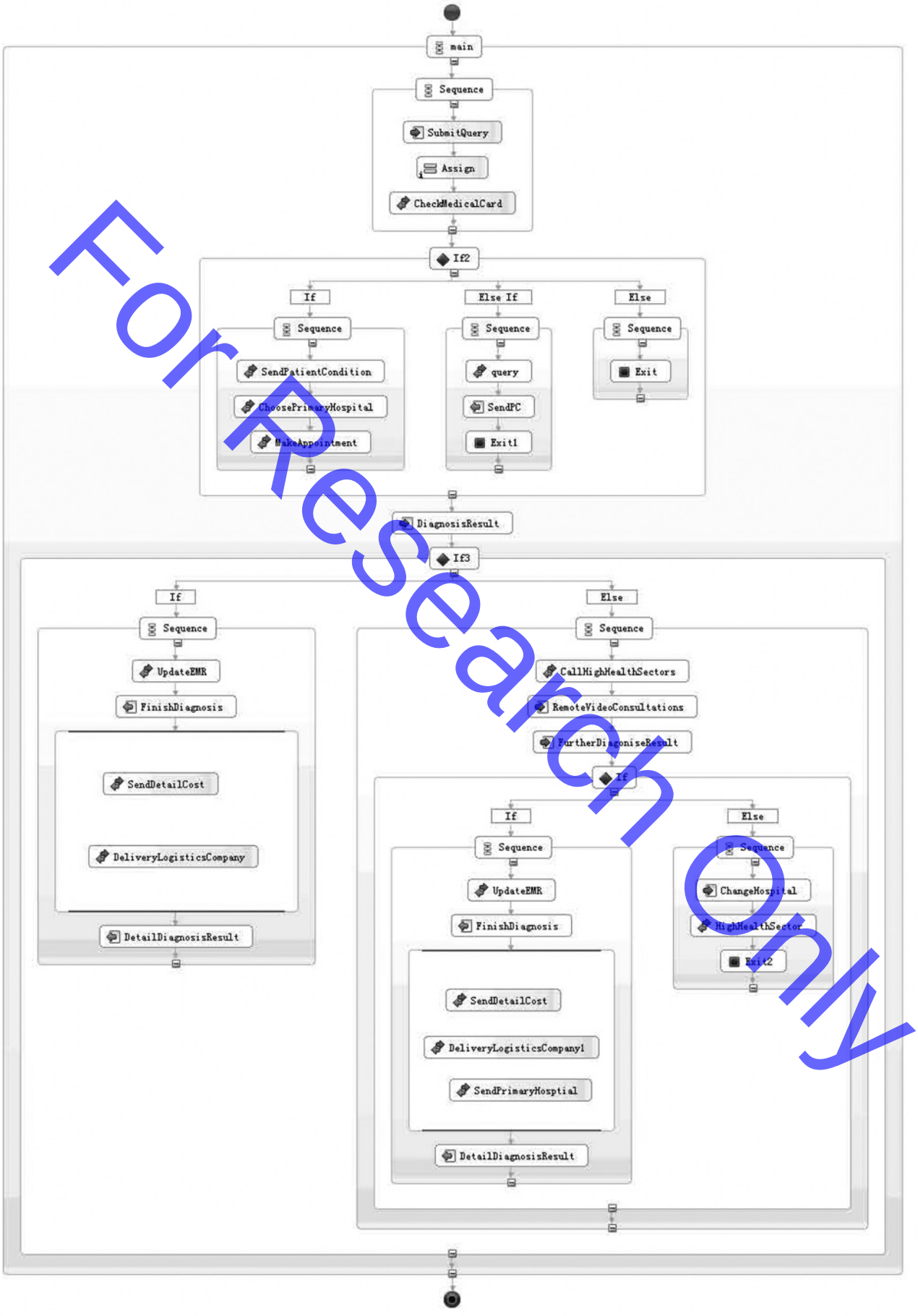
3.1 Property sequence chart

3.1.1 Introduction: PSC is an extended graphical notation of a subset of UML 2.0 SD proposed in [10]. Its design rationale is balancing 'expressive power' and 'simplicity of use'. Fig. 2 shows the PSC graphical elements. A PSC specification has sets of component instances, messages, constraints and operators. Two basic message types are available: *arrowMSGs* and *intraMSGs*. The *arrowMSGs* have three subtypes: Regular, Required and Fail. Regular messages (labelled with *e:msg*) are used to define the precondition for a desired (or an undesired) interaction. Required messages (labelled with *r:msg*) must be exchanged by the system and are used to express mandatory interactions. Fail messages (labelled with *f:msg*) should never be exchanged and are used to express undesired interactions. *IntraMSGs* are used to describe constraints that restrict the exchange of messages (*arrowMSGs*). Constraints are classified into two categories: unwanted message constraints and chain constraints. An unwanted message constraint is specified for a set of *intraMSGs* that the system must not exchange. In other words, an unwanted message constraint describes the event(s) or interactions that are disallowed between two component instances. Chain constraints are defined as a sequence of dependent *intraMSGs*, and are further classified as wanted and unwanted. Wanted chain constraints are satisfied if the

For Research Only

© The Institution of Engineering and Technology 2013

doi: 10.1049/iet-sen.2012.0047



For Research Only

Fig. 1 *Main BPEL structure for OMA example*

© The Institution of Engineering and Technology 2013

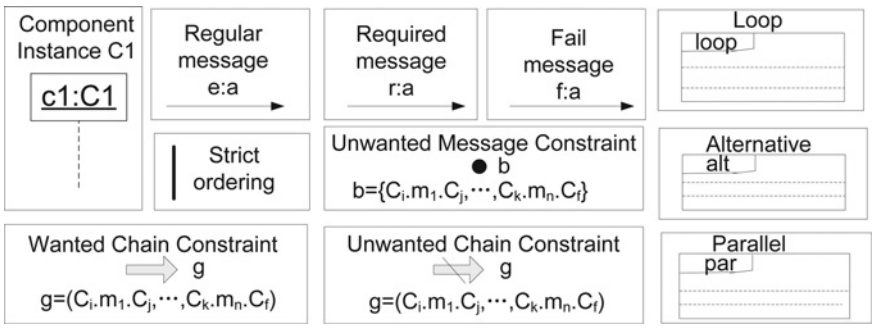


Fig. 2 *PSC graphical notations*

messages are exchanged following the sequence imposed by the chain specifications. Unwanted chain constraints require that the messages do not occur in the sequence specified in the chain specification.

Constraints are also classified into past constraints and future constraints. Past constraints specify message exchanges, wanted or unwanted, before a specific message exchange event takes place, and future constraints specify the constraints afterwards. Graphically, past constraints are located near the arrow source and future constraints are located near the arrow target of an *arrowMSG*.

PSC has five operators: loose, strict, parallel, loop and alt, which define how *arrowMSGs* can be composed. The loose operator is the default operator that defines the order of messages. However, any other messages can occur between these messages. The strict operator explicitly specifies a sequential ordering between a pair of messages, and no other message is allowed between them. The parallel, loop and alt operators specify parallel merging (i.e. interleaving), iteration and alternative behaviour, respectively.

3.1.2 Semantics: The original operational semantics of PSC is given by Büchi automata [10]. The algorithm Psc2Ba defined in [10] translates PSCs into corresponding Büchi automata. However, Büchi automata can only accept infinite traces and is not suitable for monitoring purpose since run-time traces are finite. Consequently, this paper will use another semantics domain called finite automata (FA) which can accept finite traces. We make use of basic translation rules and compositional rules to translate PSCs into FAs. The basic translation rules are used for directly deriving a FA corresponding to a single **arrowMSG** (Regular, Required or Fail messages). Then, compositional rules are applied to PSC operators: **Par**, **Loop** and **Alt**. Furthermore, we give a trace-based denotational semantics of PSC in Section 8.1. The correctness of operational semantics can be proved by demonstrating the consistency between the two semantics.

The formal definition of FA for basic rules is defined first.

Definition 1 (FA for basic rules): A FA generated from a basic rule is a six-tuple $FA\_b = \langle S, \Sigma, s_0, T, G, A \rangle$, where

- S is the finite set of states;
- Σ is the message alphabet exchanged among different services;
- $s_0 \in S$ is the initial state;
- $T \subseteq S \times \Sigma \times S$ is the finite transition set. $t = (s, \sigma, s') \in T$ is a transition, where $\sigma \in \Sigma$, $s \in S$ and $s' \in S$;
- $G \subseteq S$ is the set of glue states. Glue state means the current trace satisfies current PSC message and the system has a valid continuation. Glue state is used to merge the initial state of the next *FA_b* to form a more complex *FA_b*.
- $A \subseteq S$ is the set of accepting states. The *FA_b* describes the complementary behaviours of each basic message. Consequently, accepting state means the current finite trace violates the PSC specification.

In the generated *FA_b*s, other states are called intermediate state set. Intermediate states mean undecided result and need more traces to draw a conclusion.

Basic semantics for PSC: The Regular messages represent the construction of a pre-condition. If a Regular message does not happen, then the monitor does not detect failures (i.e. the property is still undecided or valid). But if a Regular message (or a set of messages) happens then a pre-condition has been satisfied and the continuation of the PSC must be explored. Therefore the generated *FA_b*s do not contain any accepting states but they contain glue states that are reached when the Regular messages are exchanged. Fig. 3 shows the basic rules for Regular messages. The *e* loose rule (ER1_PSC) represents the rule for Regular message in which if a happens then it causes a transition to a glue state. The self-transition labelled $!a$ ($!a$ is an abbreviation of message label $C_1.!a.C_2$, we will use this kind of abbreviation in the rest of the rule definition) in the state s_0 means that other messages can be exchanged before a. The rule *e* strict (ER2_PSC) is for the strict operator. Different from the *e* loose rule, after having reached the state s_0, we have a valid continuation to be explored if and only if the next exchanged message is exactly a. ER3_PSC (*e* past unwanted message) is the combination of the message a with the unwanted messages constraint b, where $b = \{m_1, m_2, \ldots, m_n\}$ is the unwanted message set. The idea is that we have a valid continuation if a happens and during its past no message $m \in b$ has been exchanged, that is, the construction of the valid continuation loses if in the past one message $m \in b$ has been exchanged. This is obtained by means of the self-loop labelled b & $!a$ at state s_0. ER4_PSC (*e* future unwanted message) is the case for the future constraint. In this case, we want to have one message $m \in b$ after having reached the valid continuation on s_1. The valid continuation is no longer valid if one message $m \in b$ is exchanged. ER5_PSC (*e* strict future unwanted message) is exactly the intuitive combination of *e* strict with *e* future. ER6_PSC (*e* past unwanted chain) is a combination of unwanted chain g and message a, where $g = (m_1, m_2, \ldots, m_n)$ is the message chain. If g does not happen completely and a is exchanged, the monitor will reach a valid continuation. ER7_PSC (*e* past wanted chain) is the wanted case. Consequently, if g happens completely and a is

For Peer Review Only

© The Institution of Engineering and Technology 2013

doi: 10.1049/iet-sen.2012.0047

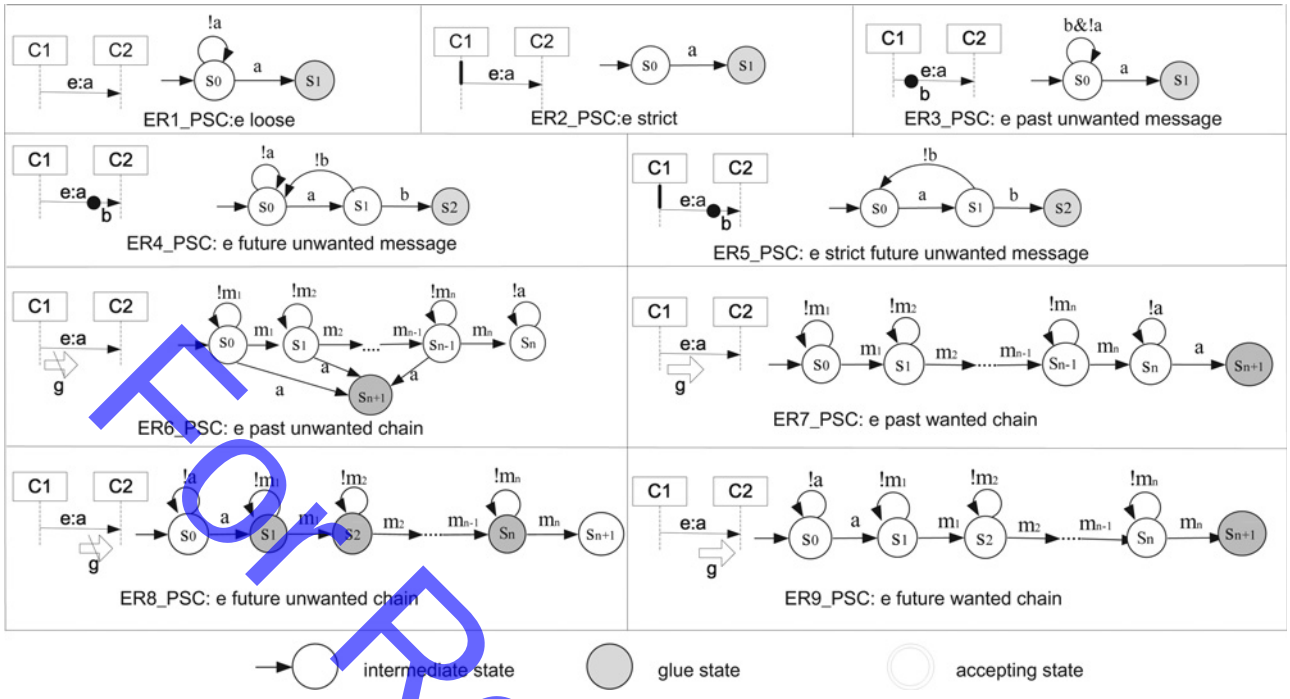


Fig. 3 *Operational semantics of Regular messages of PSC*

exchanged, the monitor will reach a valid continuation. ER8_PSC and ER9_PSC are the future cases of unwanted and wanted chains.

The semantics rules for Required message are shown in Fig. 4. A Required message is a message that must be exchanged within finite traces (in practice, we can judge whether the number of traces reaches a particular large number). In the case of RR1_PSC (*r* loose), if the message does not happen within finite traces, then the monitor will go to the accepting state and the property violation is detected. A valid continuation can be reached if *a* happens within finite traces. RR2_PSC (*r* strict) shows that if any other message but *a* (i.e. !*a* in the figure) happens while in state s_0, then the monitor goes to the accepting state immediately and there is no chance for satisfying the property. RR3_PSC (*r* past unwanted message) raises an

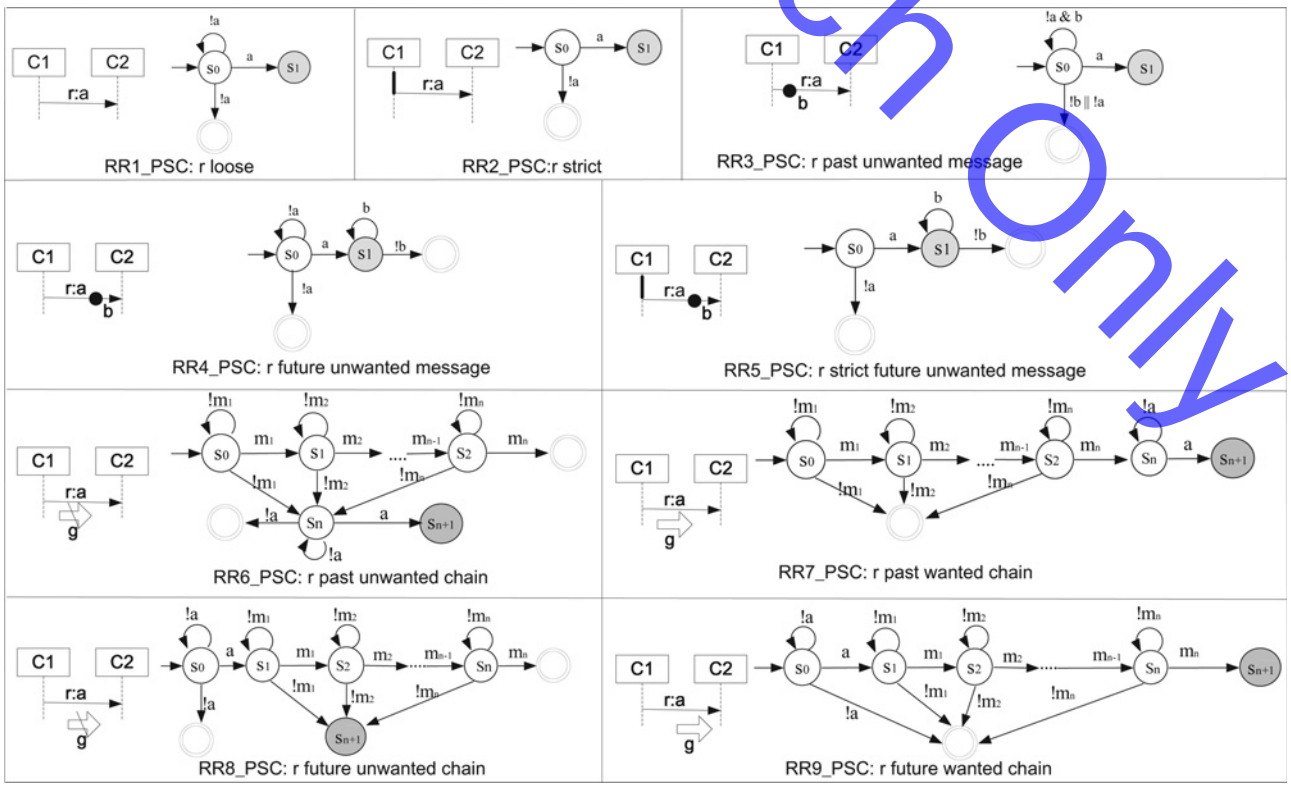


Fig. 4 *Operational semantics of Required messages of PSC*

For Research Only

© The Institution of Engineering and Technology 2013

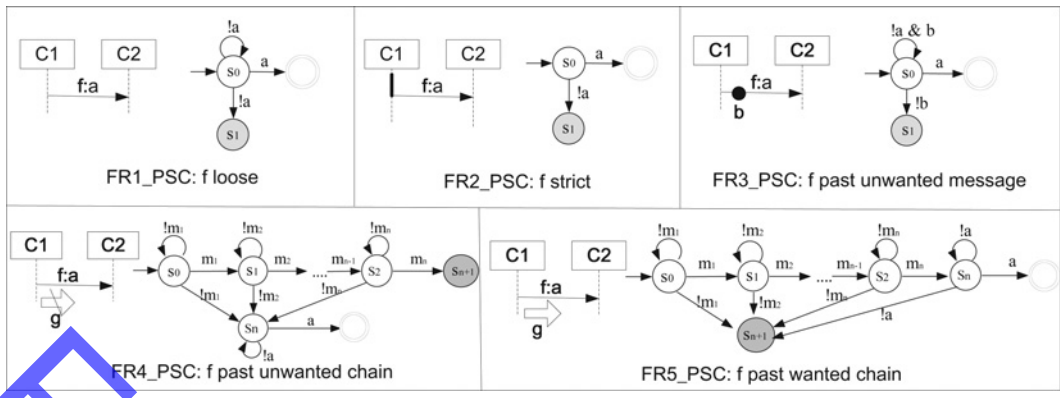


Fig. 5 *Operational semantics of Fail messages of PSC*

error if the messages in b happens or within finite traces a does not happen. As both message a and all messages in b are not exchanged, the monitor has a valid continuation. RR4_PSC (r future unwanted message) shows that if a happens and within finite traces the messages in b does not happen, the system goes to the glue state s_1. Otherwise, if a does not happen within finite traces or even in s_1 the messages in b happen, the property violation will be detected. RR5_PSC is the strict case. Consequently, there is no other message allowed before a. RR6_PSC describes the r past unwanted chain g, where $g = (m_1, m_2, \ldots, m_n)$. When the message chain g happens completely, the property violation is detected. When the message does not happen completely and a happens within finite traces, the monitor will go to the glue state. RR7_PSC is the wanted case. When the chain g happens completely and a exchanged, the monitor will go to the glue state, otherwise, a property violation will be detected. Similarly, RR8_PSC and RR9_PSC are the future unwanted and wanted cases.

A Fail message is a message that should never occur within finite traces in the system (see Fig. 5). FR1_PSC shows that if the message a does not happen within finite traces, then FA_b first has a finite cycle and then go to state s_1 (the valid continuation) and the property is not violated; exactly when the first message a happens, the FA_b reaches an accepting sink node. Considering rule FR2_PSC (*f strict*) there is an error only if a happens as the first message. FR3_PSC is the rule for Fail message with past unwanted message constraint. In particular, in this case the past constraints represent restrictions that should hold in the past in order to have a failure with the fail message. For the past constraint, if b is false before a happens then the 'precondition' for the failure is falsified. Then we do not have an undesired behaviour but we can reach the valid continuation on s_1. FR4_PSC and FR5_PSC represent the cases for past unwanted chain and wanted chain. In FR4_PSC, the monitor will have a valid continuation if and only if the chain happens completely and a does not happen. In FR5_PSC, the monitor will detect a failure if and only if the chain happens completely and a happens. Note that Fail message does not have future failures since the system will have no future after failures.

Compositional semantics for PSC: The compositional semantics for PSC are classified into two types. The first type includes two basic operators Merge and Alternative. Merge is used for sequential composition of two FA_bs. Alternative is used for alternative composition of two FA_bs. The second type includes **Alt**, **Par** and **Loop**. These operators can be defined based on basic operators **Merge** and **Alternative**.

Definition 2 (Merge for two FA_bs): Given two FA_bs generated by basic rules, $FA = \langle S, \Sigma, s_0, T, G, A\rangle$ is the merge composition of FA_1 and FA_2, denoted as $FA = FA_1 \cdot FA_2$, where

- $FA \cdot S = FA_1 \cdot S \cup FA_2 \cdot S$
- $FA \cdot \Sigma = FA_1 \cdot \Sigma \cup FA_2 \cdot \Sigma$
- $FA \cdot s_0 = FA_1 \cdot s_0$
- $FA \cdot T = FA_1 \cdot T \cup FA_2 \cdot T \cup T_{\text{glue}}$, where $T_{\text{glue}} = \{t | t \cdot s \in FA_1 \cdot G \wedge t.s' = FA_2 \cdot s_0 \wedge t \cdot \sigma = \varepsilon\}$, T_{glue} adds an empty(ε) transition from each glue state of FA_1 to the initial state of FA_2
- $FA \cdot G = FA_2 \cdot G$
- $FA \cdot A = FA_1 \cdot A \cup FA_2 \cdot A$

Definition 3 (Alternative for two FA_bs): Given two FA_bs generated by basic rules, $FA = \langle S, \Sigma, s_0, T, G, A\rangle$ is the alternative composition of FA_1 and FA_2, denoted as $FA = FA_1 \otimes FA_2$, where

- $FA \cdot S = FA_1 \cdot S \cup FA_2 \cdot S \cup \{s_0\}$
- $FA \cdot \Sigma = FA_1 \cdot \Sigma \cup FA_2 \cdot \Sigma$
- $FA \cdot s_0 = s_0$
- $FA \cdot T = FA_1 \cdot T \cup FA_2 \cdot T \cup \{(s_0, \varepsilon, FA_1 \cdot s_0)\} \cup \{(s_0, \varepsilon, FA_2 \cdot s_0)\}$
- $FA \cdot G = FA_1 \cdot G \cup FA_2 \cdot G$
- $FA \cdot A = FA_1 \cdot A \cup FA_2 \cdot A$

The *Merge* and *Alternative* operators will generate new FAs with $\varepsilon-$ transitions. However, an equivalent FA without $\varepsilon-$ transitions can be obtained through standard automata operations [15]. Furthermore, they can be extended to a finite number of FA_bs.

Definition 4 (Merge for n FA_bs): Given n FA_bs generated by basic rules, $FA = \langle S, \Sigma, s_0, T, G, A\rangle$ is the merge composition of n basic FA_bs.

- $FA_1 \cdot FA_2 \cdot FA_3 \cdot \ldots \cdot FA_n = (((FA_1 \cdot FA_2) \cdot FA_3) \ldots \cdot FA_n)$

Definition 5 (Alternative for n FA_bs): Given n FA_bs generated by basic rules, $FA = \langle S, \Sigma, s_0, T, G, A\rangle$ is the alternative composition of n basic FA_bs.

For Research Only

© The Institution of Engineering and Technology 2013

doi: 10.1049/iet-sen.2012.0047

• $FA_1 \otimes FA_2 \otimes FA_3 \otimes \cdots \otimes FA_n = (((FA_1 \otimes FA_2) \otimes FA_3) \cdots \otimes FA_n)$

Based on operators **Merge** and **Alternative**, the operators of **Alt**, **Par** and **Loop** can be defined as follows:

Definition 6 (Alt operator of PSC): $\mathrm{Alt}(\omega^{i_1,j_1}, \omega^{i_2,j_2}, \ldots, \omega^{i_r,j_r}, r)$ has r parts of selections.

Definition 7 (Par operator of PSC): $\mathrm{Par}(\omega^{i_1,j_1}, \omega^{i_2,j_2}, \ldots, \omega^{i_r,j_r}, r)$ has r parts of selections. Owing to combinatorics, the number of message sequences generated by parallel composition is $\mathrm{num(Par)} = ((n!)/(k_1!k_2! \ldots k_r!))$, where k_1, k_2, …, k_r are the numbers of these r parts of messages.

• $\mathrm{Par}(\omega^{i_1,j_1}, \omega^{i_2,j_2}, \ldots, \omega^{i_r,j_r}, r) = \omega^{i_1,j_1} \otimes \omega^{i_2,j_2} \otimes \cdots \otimes \omega^{i_{\mathrm{num(Par)}},j_{\mathrm{num(Par)}}}$, where $\omega^{i_k,j_k} = \mathrm{msg}_{i_k} \cdot \mathrm{msg}_{i_{k+1}} \cdot \ldots \cdot \mathrm{msg}_{j_k}$ $(1 \le k \le \mathrm{num(Par)})$

Definition 8 (Loop operator of PSC): $\mathrm{Loop}(\omega^{i,j}, m, n)$, the upper bound of loop is n and lower bound of loop is m.

• $\mathrm{Loop}(\omega^{i,j}, m, n) = \omega^{i,j} \otimes 2\omega^{i,j} \otimes \cdots \otimes (m-n+1)\omega^{i,j}$, where $\omega^{i,j} = \mathrm{msg}_i \cdot \mathrm{msg}_{i+1} \cdot \ldots \cdot \mathrm{msg}_j$ and $k\omega^{i,j} = (\omega^{i,j} \cdot \omega^{i,j} \cdot \ldots \cdot \omega^{i,j}/(k\,\mathrm{times}))$ $(1 \le k \le m - n + 1)$

Definition 9 (FA for whole PSC): A FA generated from the whole PSC is a six-tuple $FA\_w = \langle S, \Sigma, s_0, T, F, A \rangle$, with the different from FA_b being that the glue state set G is replaced by the final state set F.

3.1.3 Expressiveness: PSC can describe temporal properties of systems, that is, specifying required message to represent liveness properties, or using fail message to represent safety properties. In order to use PSC efficiently, the expressiveness of PSC has been evaluated by PSPs proposed by Dwyer *et al.* [6].

PSPs are used to easily represent system requirements. PSPs are divided into occurrence patterns and order patterns. Occurrence patterns specify that a given event occurs during the execution of system, which contain Absence, Existence, Bounded Existence and Universality. Order patterns specify several events to occur in sequence during system execution, which contain Precedence, Precedence Chain, Response and Response Chain. Each pattern is associated with five scopes that represent the execution regions of system when the pattern must hold. The five basic scopes are globally, before, after, between and, and after until, respectively. Autili *et al.* have already used PSCs to represent all the PSPs, which can be found in the website http://www.di.univaq.it/psc2ba/patternsMapping.php.

3.2 Timed property sequence chart

3.2.1 Introduction: In UML SD, time stamps can be attached to messages. Thus, constraints on a lower and an upper time bound between two continuous messages can be formulated. Similar to the messages in UML SD, each message in PSC can also be annotated with time constraints, or time constructs for short.

As shown in Fig. 6*a*, a regular message ***e*****:** *a* is extended as ***e*****:** *a*; $x < t$, $y := 0$, which means that ***e*****:** *a* is expected to happen before t, and then a clock y is reset. Since it is a regular message, the system will not raise an error if ***e*****:** *a* does not happen within the desired time constraint. However, when ***e*****:** *a* is replaced by ***r*****:** *a*, it means a required message ***r*****:** *a* must be exchanged before time t. If it does not happen, the system will raise an error. When ***e*****:** *a* is replaced by ***f*****:** *a*, it means that the system will go to an error state when ***f*****:** *a* is exchanged before t.

3.2.2 Semantics: In [11], we have defined the formal translational semantics that maps a TPSC specification into a corresponding timed Büchi automaton (TBA). TBA can only accept infinite traces. Similar to PSC, we also use another formalism called TA as the corresponding semantics domain. We can also define the time trace-based denotational semantics of TPSC. The correctness of the translating rules can be proved by ensuring the consistency between the two semantics.

Definition 10 (Clock constraints): For a set of clocks X, a clock constraint δ from the set of clock constraints $\Phi(X)$ can be defined as follows [16, 17]

$$\delta := x < c | x \le c | x > c | x \ge c | \neg\delta | \delta_1 \wedge \delta_2$$

where $x \in X$ is a clock variable, and $c \in \mathbb{N}$ is a constant.

Definition 11 (Evaluation functions): Several evaluation functions have been defined. The function $\vDash$ evaluates each value $v(x)$ of a clock x and checks if the value fulfils the constraint δ. The function $[\delta]^{\mathrm{val}} = \{v(x) | v(x) \vDash \delta\}$ denotes all the values of a clock x which satisfy δ. We assume that a clock constraint δ is homogenous, meaning that the clock constraint is fulfilled only for a single connected set of clock values. Two additional functions are also defined: $[\mathrm{pre}(\delta)] = \{v(x) | v(x) < \min([\delta(x)])\}$ and $[\mathrm{succ}(\delta)] = \{v(x) | v(x) > \max([\delta(x)])\}$, which respectively describe all clock values

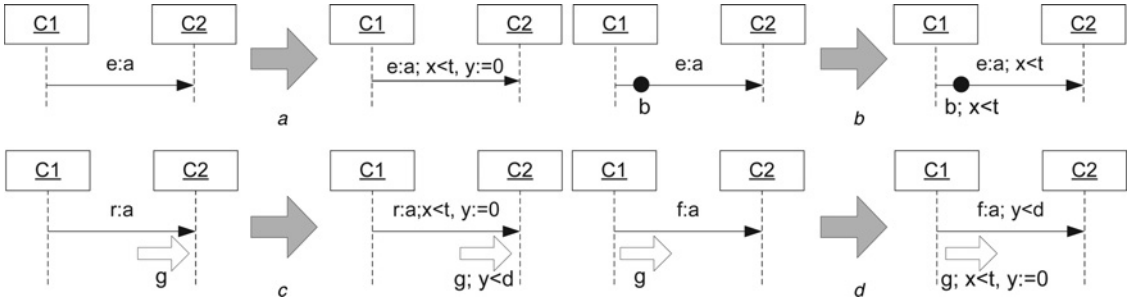


Fig. 6 *Extending PSC with time constructs*

For Research Only

© The Institution of Engineering and Technology 2013

that do not fulfil the clock constraint and happen before and after the clock constraint.

For example, if $\delta(x) = x \geq 2 \wedge x \leq 4$, $[\text{pre}(\delta)] = \{v(x)|v(x) < 2\}$, and $[\text{succ}(\delta(x))] = \{v(x)|v(x) > 4\}$.

Definition 12 (Clock reset): A clock reset for a clock x is defined as $v(x) := 0$. Normally, we denote ψ as a set of clock reset.

The formal definition of TA is defined first.

Definition 13 (TA for basic rules): A TA generated from a basic rule of TPSC is a seven-tuple $TA = \langle \Sigma, S, s_0, F, G, \text{Clock}, T \rangle$, where

- $\Sigma \cup 1$ is a finite set of simple message labels exchanged in the system, where '1' means that any messages can happen.
- S is a finite set of states;
- $s_0 \in S$ is the initial state;
- $F \subseteq S$ is the finite set of accepting states. Note that TA also represents the complementary behaviours of TPSC, so the accepting state represents a failure of the system;
- $G \subseteq S$ is the set of glue states. They connect the initial state of another TA to form a new TA;
- Clock is a finite set of clocks, as defined in Definition 10;
- $T: S \times S \times \Sigma \times \Phi(\text{Clock}) \times 2^{\text{Clock}}$ is a set of transitions. $\Phi(\text{Clock})$ is a finite set of clock constraints. A transition $t \in T$ is denoted as $t = \langle s, s', a, \delta, \psi \rangle$, which defines state transition from s to s' with message $a \in \Sigma \cup 1$, the clock constraint $\delta \in \Phi(\text{Clock})$, and the set of clock reset $\psi \subseteq \text{Clock}$.

In the following, we will define the formal translational semantics that maps a TPSC specification into a corresponding TA. The rules are categorised into basic and compositional rules. Basic rules identify how to translate a single message in a TPSC specification into a TA, whereas compositional rules show how to compose these basic automata according to structured operators, such as *par*, *loop* and *alt*.

Basic rules for TPSC: Fig. 7 shows the corresponding rules for generating basic TAs for Regular messages. The translation rules are similar to the rules of PSC, with each translation denoted by clock constraint or clock reset. The symbols δ, δ' and δ'' are used to represent clock constraints for messages, past constraints and future constraints, respectively, as defined in Definition 10. ψ, ψ' and ψ'' represent the sets of reset clocks for messages, past constraints and future constraints, respectively, as defined in Definition 12. ER1_TPSC(*e loose*) rule shows if a is not exchanged and other messages happen, the TA stays at s_0. When a happens within δ, the TA will go to the glue state s_{glue} with clock reset ψ. ER2_TPSC rule is the strict case. In ER3_TPSC rule, when the messages in b do not happen within δ' and a does not happen within δ, the TA stays at s_0. Otherwise, the TA will go to the glue state if a happens in line with δ. ER4_TPSC is the future case, if a happens under δ the TA will go to the glue state that requires the messages in b do not happen under δ''. ER5_TPSC is the strict case of future unwanted message. In ER6_TPSC, when the chain g does not happen completely under δ' and a happens under δ, the TA will go to the glue state. In ER7_TPSC, if g happens completely satisfying δ' and a happens satisfying δ, the TA will go to the glue state. In ER8_TPSC, when some messages in g happens under pre (δ'') or g has not happened completely until succ(δ''), the TA will go to the glue state. In ER9_TPSC, if a happens satisfying δ and g happens completely, the TA will go to the glue state.

Fig. 8 shows the rules for Required messages of TPSC. RR1_TPSC corresponds to a Required message a with a clock constraint δ. The difference from ER1_TPSC is that when a happens before pre(δ) or a has not happened until succ(δ), the TA will go to the accepting state. RR2_TPSC is the strict case. Consequently, the difference is that no other messages are allowed before a within δ. RR3_TPSC is the case with past unwanted message, when the messages in b happen and fulfil δ', the TA will go to the accepting state. When the messages in b do not happen according to δ', and a happens and fulfils δ, TA will have a valid

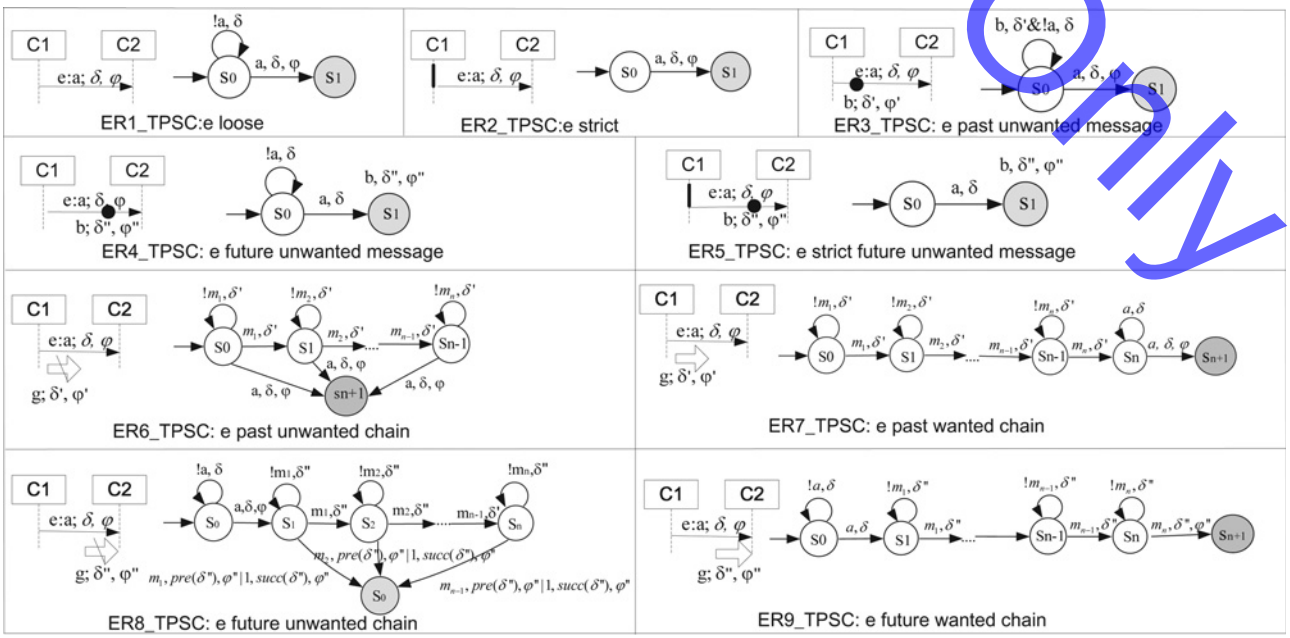


Fig. 7 *TAs for Regular messages of TPSC*

For Review Only

© The Institution of Engineering and Technology 2013

doi: 10.1049/iet-sen.2012.0047

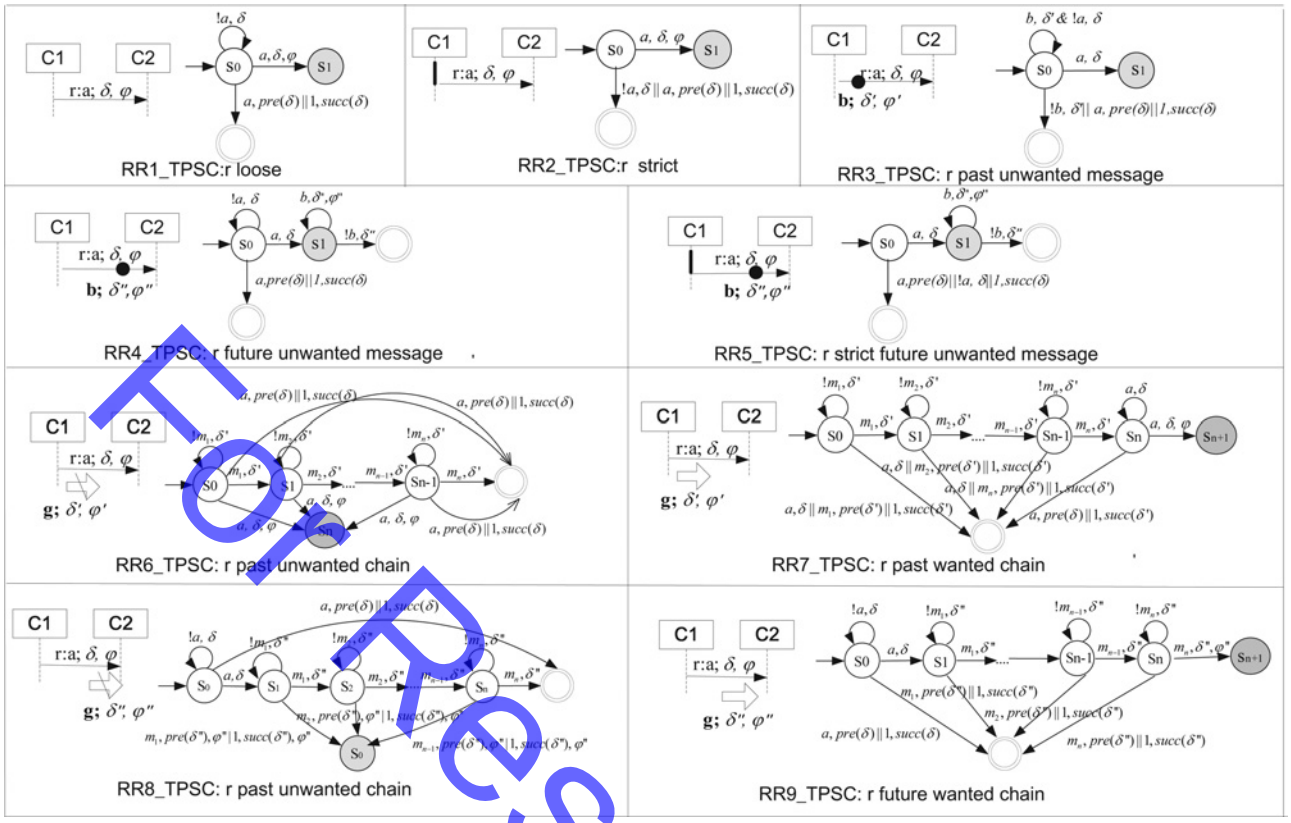


Fig. 8 *TAs for Required messages of TPSC*

For Review Only

continuation. In RR4_TPSC, when a happens within δ, the TA will go to the glue state. Then the TA also requires that the messages in b have not happened under δ'' at the glue state. RR5_TPSC is the strict case, with the difference from RR4_TPSC being that no other messages are allowed before a within δ. RR6_TPSC is the case with past unwanted chain. If g does not happen completely under δ', and a is exchanged under δ, TA will go to the glue state. Otherwise when g happens completely under δ', or g does not happen completely but a does not happen under δ, the TA will go to the accepting state and an error is raised. In RR7_TPSC, if g happens completely and a happens under δ, the TA will go to the glue state. In all other cases, the TA will raise an error. RR8_TPSC, when a happens under δ and g has happened completely under δ'', the TA will go to the accepting state. Otherwise TA will go to the glue state. The only difference between Rule RR9_TPSC and RR8_TPSC is that g is expected to happen completely under δ''.

The rules for fail messages of TPSC can be defined in a similar way. As there are only two differences for rules of Fail message with respect to the Required rules, we skip the detailed presentation of these rules. First, the semantics of a Fail message is inverse to the semantics of a Required messages, in that glue states and accepting states are interchanged. Secondly, since other messages cannot follow a fail message, there is no clock reset in final transitions of the generated TAs.

Compositional rules for TPSC: Using these basic rules, basic messages in a TPSC specification can be translated into the corresponding simple TAs. However, the real-time property represented by a TPSC specification is usually composed of many basic messages with structured operators, such as *alt*, *par* and *loop*. Therefore, compositional rules are also need to be defined to compose simple TAs according to the structure of complex TPSCs. There are two kinds of compositional rules. One is to merge TAs in a sequential and alternative way, that is, *Merge* and *Alternative*. The other includes *alt*, *par* and *loop* structured operators. These operators can be defined in a similar way as PSC.

3.2.3 Expressiveness: The expressiveness of the TPSC specification language has been tested with the real-time specification patterns proposed by Konrad and Cheng [13]. These real-time specification patterns, which are a timed extension of the un-timed specification patterns [6], are proposed to support quantitative reasoning about time. The patterns are classified into duration (minimum duration, maximum duration), periodic (bounded recurrence) and real-time order (bounded response, bounded invariance). Each real-time specification pattern can be further associated with five scopes (*globally, before, after, between and,* and *after until*) as defined in [6]. In our previous work, we have already used TPSC to represent all these patterns [11].

3.3 Probabilistic timed property sequence chart (PTPSC)

3.3.1 Introduction: According to the same design rationale of PSC and TPSC, we later designed a new probabilistic scenario-based specification called PTPSC [12]. Each message or operator can be annotated with a probability. Furthermore, according to the idea of LSCs [9], we add the pre-chart to PTPSC to describe a trigger condition for starting a monitoring run. The messages in the pre-chart are restricted to the *regular* type. Following the

© The Institution of Engineering and Technology 2013

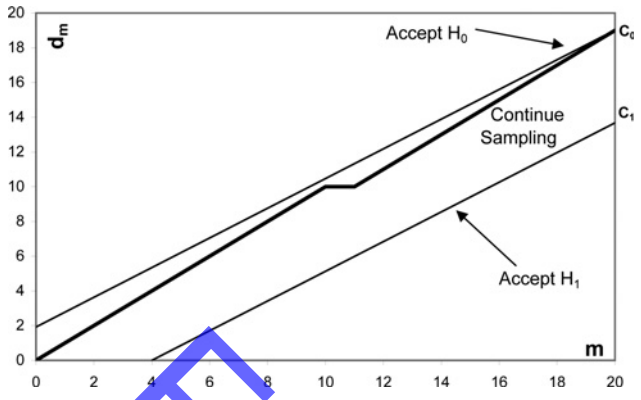


Fig. 9 *Graphical interpretation of the SPRT*

pre-chart in PTPSC, a main chart is enriched with a probability operator. PTPSC only supports one probability construct. However, if a probabilistic property has more than one probability, it can be modelled by a joint probability of multiple messages.

3.3.2 Semantics: One challenge of defining the formal semantics of PTPSC specifications is that there is no clear accepting or rejecting condition like that of traditional temporal or timing specifications. To overcome this problem, we propose to use sequential statistical hypothesis testing based on a null hypothesis (H_0 the system fulfils the probabilistic property) and an alternative hypothesis (H_1 the system does not fulfil the probabilistic property). The specific statistical hypothesis test we use is the SPRT [18] that bounds the probabilities of false accepting or rejecting the null hypothesis to two test parameters α and β, respectively. Consequently, the operational semantics of PTPSC is defined by two parts:

- Based on the defined syntax, we can translate a PTPSC property into a TA using a syntax directed translator (SDT) [19]. The translation rules of SDT are also categorised into basic and compositional rules [11].
- We design an algorithm to do sequential hypothesis testing. The inputs of this algorithm are the timed automata (TA) for PTPSC, two hypothesis H_0 and H_1 that are defined based on the probability p specified in the PTPSC and other hypothesis parameters such as α and β. The outcome of the hypothesis test is that H_0 is accepted, or H_1 is accepted, or undecided, implying more samples are needed. In practise, as shown in Fig. 9 the test procedure compares the number of correct samples d_m of m experiments with two functions $c_0(m)$ and $c_1(m)$. If $d_m > c_0(m)$, then H_0 is accepted. If $d_m < c_1(m)$, then H_1 is accepted. Otherwise, when $c_0(m) > d_m > c_1(m)$, it cannot be decided and more samples are needed.

3.3.3 Expressiveness: PTPSC is designed to represent probabilistic properties. In order to measure the expressiveness of the PTPSC specification language, we test it with respect to the PSP system ProProST (Probabilistic Property Specification Templates) of common probabilistic properties proposed by Grunske [14]. The specification pattern system contains eight generic patterns. PTPSC can represent seven patterns including the Transient State Probability pattern, the Probabilistic Invariance pattern, the Probabilistic Existence pattern, the Probabilistic Precedence pattern, the Probabilistic Response pattern, the Probabilistic Constrained Response pattern, and Steady State Probability [4]. Note that the PSP Steady State Probability requires long running behavioural analysis of the system and cannot be used for monitoring. Consequently, this pattern cannot be expressed with PTPSC.

3.4 Example

The informal requirements of the running example can be represented by PSC, TPSC and PTPSC specifications, corresponding to temporal, timing and probabilistic properties, respectively.

For $Prop_1$ to $Prop_3$, PSC specifications can be used to represent them. As an example, Fig. 10 shows the PSC specification of $Prop_1$. It is composed of four messages: a Regular message, a Required message with future unwanted chain constraint and two Required messages. Using the translating rules in basic rules (ER1_PSC, RR4_PSC, RR1_PSC and RR1_PSC) and compositional rules (**Merge**), the generated FA for $Prop_1$ is shown in Fig. 11. Within finite traces, the monitor can stay on S_0 to S_4 and

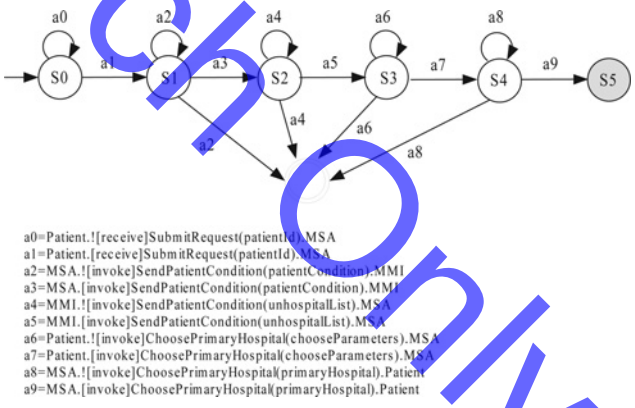


Fig. 11 *Generated FA for $Prop_1$*

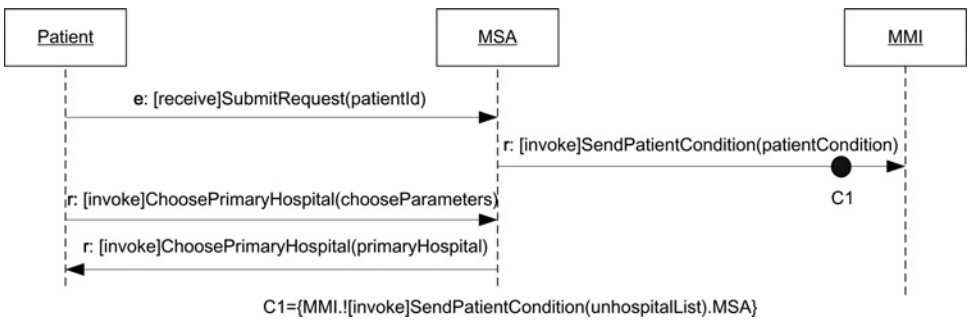


Fig. 10 *PSC specification for $Prop_1$*

For Research Only

© The Institution of Engineering and Technology 2013

doi: 10.1049/iet-sen.2012.0047

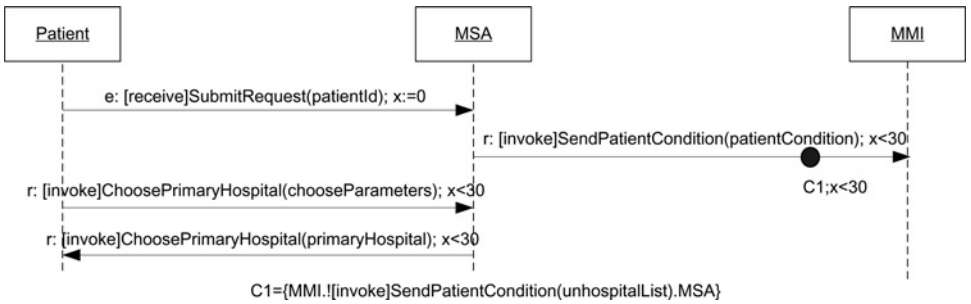


Fig. 12 *TPSC specification for $Prop_4$*

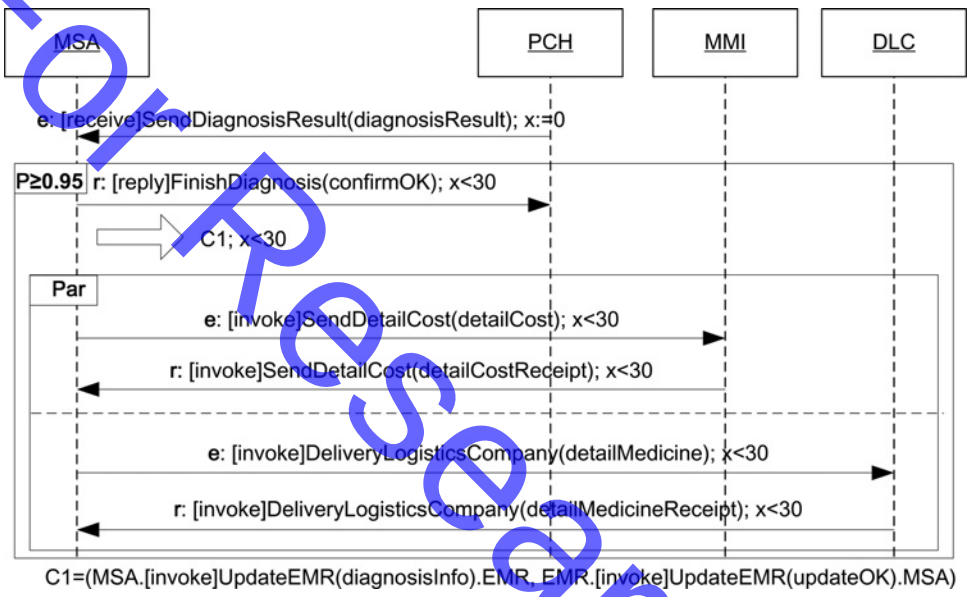


Fig. 13 *PTPSC specification for $Prop_7$*

For Research Only

eventually goes to final state S_5 when the desired messages happen. However, if $a3$, $a5$, $a7$ and $a9$ do not happen within finite traces, the system will raise failures.

For $Prop_4$ and $Prop_5$, TPSC specifications are used to represent them. Fig. 12 shows the TPSC specifications of $Prop_4$ as an example. The structure of this property is similar to $Prop_1$ except adding some timing information.

Fig. 13 shows the PTPSC specifications of $Prop_7$ as an example. It is composed of a total of six messages: first, a Regular message and a Required message with past chain constraint. Then there are two parallel parts, each with a Regular message and a Required message. After adding the time reset clock and time constraints to each message, the first message is a trigger message, and all the messages in the main chart may happen with a probability of 95%.

4 WS-PSC tool

This section first shows the framework of WS-PSC monitor and then shows how to monitor temporal, timing and probabilistic properties in WS-BPEL process with our running example.

4.1 Framework overview

The framework of the prototype tool WS-PSC monitor is illustrated in Fig. 14, consisting of four main components Interceptor, Observer, Translator and Analyzer, respectively.

- *Interceptor:* Aspect-oriented programming (AOP)-based approach [20] is used to extend the existing BPEL execution. We consider business processes as core concerns and the monitored messages as cross-cutting concerns. The messages that describe the complex interaction between

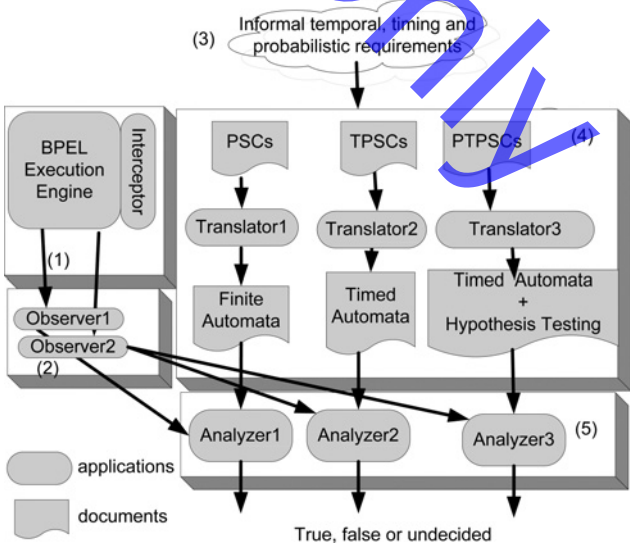


Fig. 14 *Framework of tool chain WS-PSC monitor*

© The Institution of Engineering and Technology 2013

several service components are related to these service components in the composite service. The same monitoring codes need to be repeatedly added into BPEL business processes for the corresponding service components. It results in lowering the maintainability and reusability of codes because the monitoring codes intertwined with business processes. Monitoring aspects of a composite service can be implemented with AspectJ, which is a seamless extension of Java language. It has its own constructs, such as aspects, join point, point cut and advice, etc. A joint point refers to a well-defined point during program execution. The set of joint points are called pointcut. Similar to methods, advices attached to pointcut define the extra behaviour of pointcut. Aspects are the implementation of modularity unit of cross-cutting, which include point cut, advice and type declaration. According to the identified joint points, the aspects and goal application are weaving together. The weaving process may be static (at compilation) or dynamic (at run-time). We defined a set of joint points of certain monitoring aspects, namely, the combined points between monitoring aspects and business logic are defined. A pointcut similar to Boolean operator in programming language selects a certain joint point to execute monitoring aspects. The combination of several pointcuts can observe some kinds of different joint points, which supports monitoring several service states when the business processes concurrently execute.

According to user's requirements, PSC, TPSC and PTPSC specifications are used to represent the monitored properties. Then the monitoring aspects are defined to intercept the interaction messages between basic services such as 'invoke', 'receive' and 'reply'. Finally, the defined aspects and original services are weaved together to intercept the run-time messages of composite services and add corresponding time stamps to each message.

• *Observer:* Observer collects the run-time messages intercepted by Interceptor. Usually, we can only observe SOAP messages that are related to activities. Message occurrence means activity occurrence. Global observer or local observer is used to observe message exchange or activity occurrence. Global observer can observe activities and messages at process level, concerning several services. Local observer can only observe activities or messages relative to one individual service. We focus on locally observing messages sent or received by individual service. Two observers are designed to intercept messages in PSC and TPSC, respectively. Observer1 can only collect the messages and record them in order to further check whether these messages satisfy PSC properties. Observer2 can not only collect the messages but also record the corresponding time stamps to further check these messages against TPSC and PTPSC properties.

• *Translator:* Translator is used to translate PSC, TPSC and PTPSC specifications into formats that can be accepted by Analyzer. We designed three translators for these specifications. Translator1 can translate PSC specifications into semantics domain FA. Translator2 can translate TPSC specifications into the semantics domain TA. Translator3 can translate PTPSC specification into semantics domain TA and a SPRT process.

• *Analyzer:* On receiving the messages from the Observer, the Analyzer has the ability to verify at run-time whether a property has been satisfied or not. We designed three different analysers for PSC, TPSC and PTPSC, respectively. The inputs of these algorithms are intercepted messages and the corresponding semantics domain, FA, TA and TA with SPRT process. Similar to the approach in [21, 22], we use the three-valued semantics for the monitoring outputs: 'true, false or undecided'. Note that in the generated FA or TA, there are three kinds of states: accepting state, final state and internal state. Since FA or TA describes complementary behaviours of PSC or TPSC properties, accepting state means property violation, final state means property satisfied while intermediate state means undecided.

Analyzer1 improves FA-based analysing algorithm proposed in [23]. In [23] we obtain the monitoring result according to a message set. In this paper, we obtain the monitoring result based on each intercepted message since we need to draw conclusion immediately. The main logic of Analyzer1 is shown in Fig. 15*a*. For each intercepted message *m*, Lines 1–2 show that Analyzer1 will go to next state if *m* matches the corresponding transitions in *FA*.

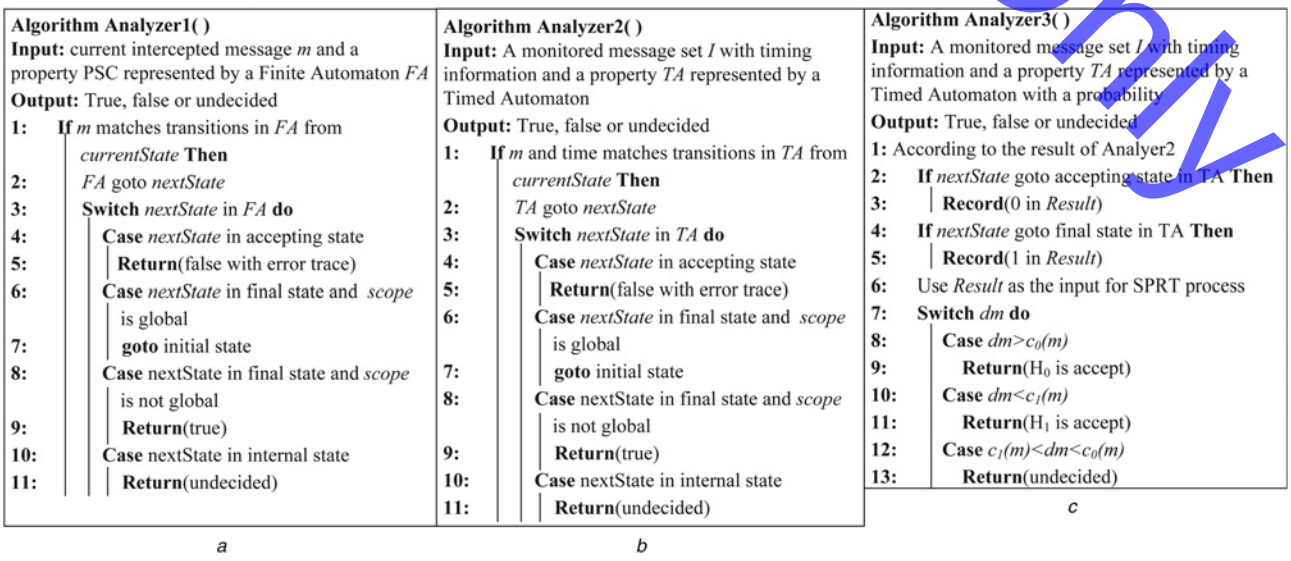


Fig. 15 *Three analysing algorithms*

For Research Only

© The Institution of Engineering and Technology 2013

doi: 10.1049/iet-sen.2012.0047

Lines 3–11 shows four cases during continuous monitoring: if *FA* goes to an accepting state, an error trace will be detected; if *FA* goes to the final state and the property *scope* is global, it will restart at the initial state again; if *FA* goes to the final state and the property *scope* is not global, it will return true; if *FA* goes to an intermediate state, it will return undecided.

Analyzer2 improves TA-based trace analysing algorithm proposed in [24]. The main idea of Analyzer2 is shown in Fig. 15*b*. The main difference between Analyzer1 and Analyzer2 is that Analyzer2 considers timing information. Consequently, it can not only analyse the message but also check that the timing stamps satisfy the transitions in TA.

Analyzer3 implements a monitor combining TA and SPRT process proposed in [4]. The main logic of Analyzer3 is shown in Fig. 15*c*. According to the result of Analyer2, if *nextState* goes to an accepting state, a '0' is recorded into *Result*; if *nextState* goes to the final state, a '1' is recorded into *Result*. Then, *Result* is used as input to the SPRT process according to the semantics of PTPSC. Finally, the checking results are returned based on lines 7–13.

The flow of the WS-PSC monitor framework is also shown in Fig. 14 and divided into the following steps:

1. AOP-based approach [20] is used to extend ActiveBPEL engine with Interceptor. Interceptor that is the core component in our framework allows for defining monitor-related pointcuts and advices in AspectJ. Our approach can define pointcuts for the following events of the engine: Engine starts or stops, BPEL process construction or destruction and key activities of BPEL process such as invoke, receive and reply. For example, in order to monitor the whole activities exchanged by the composite service and the corresponding timing information, we can define the following aspect in AspectJ to record the activities:

```
1 public aspect InterceptReceive_Log{
2     pointcut log_Receive():
3     execution( * receive.* ->receive(..));
4     before ():log_Receive (){
5        Date date=new Date();
6        SimpleDateFormatdf=new
7        SimpleDateFormat("yyyy-MM-dd hh:mm");
8        Signature 8s=thisJoinPoint.getSignature();
9        MLogFile.println(df.format(date)+
         "[Monitor LOG] Entering[receive]: "+s.toString()); }
```

In lines 2 and 3 method receive () of an arbitrary class (*) found in the receive package of the ActiveBPEL engine implementation is intercepted. Since the invoke activity only has input parameters, we need to record and add the time stamps for the parameters. Lines 4–9 record the input parameters and add the corresponding timing stamps by the use of 'before' method. For activities that just have two parameters such as invoke activity, we have to use *before()* and *after()* method to intercept the input and output parameters. Note that if only PSC specifications need to be monitored, we just need to intercept the parameters of the activities without adding timing stamps. Finally, AspectJ class files are generated from monitoring aspects by the compiler. AspectJ class files and original services in ActiveBPEL engine are weaving together. During the execution of business logic, monitoring aspects are dynamically inserted into business process for run-time monitoring.

2. According to property types, the 'Observer' can classify the intercepted messages. If the property type is temporal,

```
2011-04-10 09:51 [Monitor LOG]Entering:[receive]SubmitRequest(Id67980)
2011-04-10 09:51 [Monitor LOG]Entering:[invoke]CheckMedicard(Id67980)
2011-04-10 09:51 [Monitor LOG]Exiting:[invoke]CheckMedicard(CheckResult)
2011-04-10 09:52 [Monitor LOG]Entering:[invoke]SendPatientCondition(patientCondition)
2011-04-10 09:52 [Monitor LOG]Exiting:[invoke]SendPatientCondition(hospitalList)
2011-04-10 09:53 [Monitor LOG]Entering:[invoke]ChoosePrimaryHospital(hospitalList)
2011-04-10 09:53 [Monitor LOG]Exiting:[invoke]ChoosePrimaryHospital(primaryHospital)
2011-04-10 09:55 [Monitor LOG]Entering:[invoke]MakeAppointment(detailDate)
2011-04-10 09:55 [Monitor LOG]Exiting:[invoke]MakeAppointment(recieption)
2011-04-10 11:22 [Monitor LOG]Entering:[receive]DiagnosisResult(DiagnosisResult)
2011-04-10 11:22 [Monitor LOG]Entering:[invoke]UpdateEMR(DiagnosisInfo)
2011-04-10 11:24 [Monitor LOG]Exiting:[invoke]UpdateEMR(recieption)
2011-04-10 11:28 [Monitor LOG]Entering:[reply]FinishDiagnosis(DiagnosisResult)
2011-04-10 11:30 [Monitor LOG]Entering:[invoke]DeliveryLogisticsCompanies(detailMedicine)
2011-04-10 11:50 [Monitor LOG]Exiting:[invoke]DeliveryLogisticsCompanies(detailMedicineReciept)
2011-04-10 11:30 [Monitor LOG]Entering:[invoke]SendMedicineInsurance(detailCost)
2011-04-10 11:30 [Monitor LOG]Entering:[invoke]SendMedicineInsurance(detailCostReciept)
2011-04-10 11:55 [Monitor LOG]Entering:[reply]DetailDiagnosisResult(detialDiagnosisResult)
2011-04-20 08:51 [Monitor LOG]Entering:[receive]SubmitRequest(Id67980)
2011-04-20 08:52 [Monitor LOG]Entering:[invoke]CheckMedicard(Id67980)
2011-04-20 08:52 [Monitor LOG]Exiting:[invoke]CheckMedicard(CheckResult)
2011-04-20 08:55 [Monitor LOG]Entering:[invoke]SendPatientCondition(patientCondition)
2011-04-20 08:55 [Monitor LOG]Exiting:[invoke]SendPatientCondition(hospitalList)
2011-04-20 10:22 [Monitor LOG]Entering:[invoke]ChoosePrimaryHospital(hospitalList)
2011-04-20 10:22 [Monitor LOG]Exiting:[invoke]ChoosePrimaryHospital(primaryHospital)
2011-04-20 10:24 [Monitor LOG]Entering:[invoke]MakeAppointment(detailDate)
2011-04-20 10:28 [Monitor LOG]Exiting:[invoke]MakeAppointment(recieption)
2011-04-20 10:30 [Monitor LOG]Entering:[receive]DiagnosisResult(DiagnosisResult)
2011-04-20 10:50 [Monitor LOG]Entering:[invoke]CallHighHealthSectors(DiagnosisResult)
2011-04-20 10:55 [Monitor LOG]Entering:[invoke]CallHighHealthSectors(analyzeResult)
2011-04-20 11:22 [Monitor LOG]Entering:[receive]RemoteVideoConsultations(DiagnosisResult)
2011-04-20 11:25 [Monitor LOG]Entering:[receive]FurtherDiagnosisResult(furtherDiagnosisResult)
2011-04-20 11:30 [Monitor LOG]Entering:[invoke]UpdateEMR(DiagnosisInfo)
2011-04-20 11:33 [Monitor LOG]Exiting:[invoke]UpdateEMR(recieption)
2011-04-20 11:35 [Monitor LOG]Entering:[reply]FinishDiagnosis(DiagnosisResult)
2011-04-20 11:40 [Monitor LOG]Entering:[invoke]DeliveryLogisticsCompanies(detailMedicine)
2011-04-20 11:43 [Monitor LOG]Exiting:[invoke]DeliveryLogisticsCompanies(recieption)
2011-04-20 11:45 [Monitor LOG]Entering:[invoke]SendMedicalInsurance(detailCost)
2011-04-20 11:48 [Monitor LOG]Exiting:[invoke]SendMedicalInsurance(recieption)
2011-04-10 11:53 [Monitor LOG]Entering:[invoke]SendPrimaryHospital(DiagnosisResult)
2011-04-10 11:55 [Monitor LOG]Exiting:[invoke]SendPrimaryHospital(DiagnosisResultReciept)
2011-04-20 11:50 [Monitor LOG]Entering:[reply]DetailDiagnosisResult(detialDiagnosisResult)
...
```

For Research Only

© The Institution of Engineering and Technology 2013

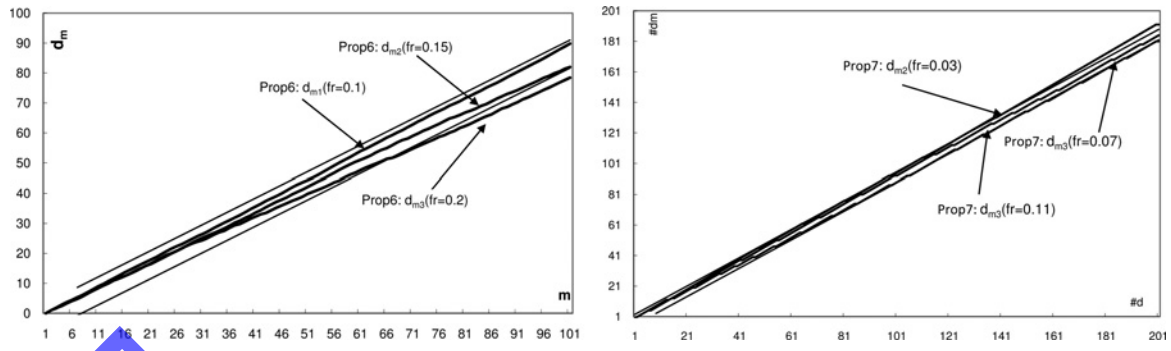


Fig. 16 *m/d_m curves for the probabilistic properties $Prop_6$ and $Prop_7$*

the Observer can only record all the interaction messages; if the property type is timing or probabilistic, the Observer needs also to record the time stamps for each interaction message. These execution messages are translated into a format that can be accepted by Analyzer. The translated execution message has the following format:

```
time stamp [Monitor LOG]{Entering/
Exiting}:[receive]/[invoke]/[reply]
messageOperation(parameter)
```

where Entering means input message, whereas Exiting means output message. An execution message sequence of OMA interacting with a patient and the corresponding time stamps are shown as follows.

3. The informal requirements are represented by PSC, TPSC and PTPSC specifications, corresponding to temporal, timing and probabilistic properties, respectively. Example property representations have been shown earlier in Section 3.4.

4. We can use the three translators of the WS-PSC monitor to translate PSC, TPSC and PTPSC into FA and TA, and a combination of TA and SPRT process, respectively. From $Prop_1$ to $Prop_3$, by using translator1, they can be translated into the corresponding FA. For example, the generated automata for $Prop_1$ is shown in Fig. 11. For $Prop_4$ and $Prop_5$, by using translator2, they can be translated into the corresponding TA. For $Prop_6$ and $Prop_7$, the generated TA are the same as those of $Prop_4$ and $Prop_5$. However, we also need to use the SPRT process to calculate the probabilities.

5. Analyzers first receive the intercepted messages and the properties represented by PSC, TPSC and PTPSC, then check whether the run-time information satisfies the desired properties. If the property type is temporal, Analyzer1 is used. If the property type is timing, Analyzer2 is used. If the property type is probabilistic, Analyzer3 is used.

Table 1 Informal descriptions and the corresponding patterns of the monitored temporal properties

Id	Informal descriptions	Patterns	Results
TA ($Prop_8$)	after sending a patient's medical parameters to the Lab and TA receives a '*high*' result, TA will immediately notify the assistants nearest to the patient	2 stimulus-2 response; scope global	yes
TA ($Prop_9$)	after sending a patients data to the Lab and the TA receives a 'needDiagnosis' result, the doctor will diagnose the patient and before that he must receive a notification from TA	precedence; scope after	yes
TA ($Prop_{10}$)	if for a certain patient the number of 'high' results is more than three times, the TA also needs to notify the doctors to diagnose the patient	3 stimulus-2 response; scope global	no
TA ($Prop_{11}$)	if the number of notifications to help a certain patient is less than five times before the end of a process, the TA should also notify doctors to diagnose the patient to see whether the patient is fine	5 stimulus-2 response; scope global	yes
OJA ($Prop_{12}$)	if the user's credit account is invalid, he (or she) cannot get a video	response; scope absence	no
OJA ($Prop_{13}$)	if the credit account of the user is detected to be valid, and he cannot get the video, there must be a message showing there is not enough money in the account	precedence; scope after	yes
OJA ($Prop_{14}$)	if there is no satisfied video in KB, VA will send query request to other providers, then some providers respond to the request. After VA provides a user with a video list (with respect to textual description of related service information) that is composed of all periodically collected responses, the user reads the video list and selects one of the providers for getting and watching the video	1 stimulus-2 response and 1 stimulus-2 response; scope global	yes
OJA ($Prop_{15}$)	when getting a video, the user can either watch the video, or she can either watch the make request to choose another video, but for the second case she have to choose the reason (e.g. the poor quality of the current video) and cancel the video	response; global or precedence chain; scope after	yes
TM ($Prop_{16}$)	if the user invokes the travel service (i.e. he/she sends the travel requirements to the TM service), TM service will respond to the user with the result of three service invocationsfindDuration, calculateTime and checkSchedule	existence; scope between and	yes
TM ($Prop_{17}$)	after the user orders the desired travel, the travel schedule will be sent to the user eventually. However, between these two messages, TA must successfully book the Hotel, Traffic and arrange a meeting with Database	existence; scope between and	yes

For Research Only

© The Institution of Engineering and Technology 2013

doi: 10.1049/iet-sen.2012.0047

Table 2 Informal descriptions and the corresponding patterns of the monitored timing properties

Id	Informal descriptions	Patterns	Results
TA ($Prop_{18}$)	after sending a patients medical parameters to the Lab, and the TA receives a 'high' result within 1 hour, the TA will immediately notify the assistants nearest to the patient and must receive a response within 1 hour	bounded 1 cause 2 response; scope after	yes
TA ($Prop_{19}$)	after sending the patient's medical parameters to the Lab, TA receives a 'high' result within 1 hour. TA will immediately notify the assistants nearest to the patient and must receive a response from them within 1 hour	bounded 1 cause 2 response; scope after	yes
TA ($Prop_{20}$)	if for a certain patient the number of 'high' results is more than three times during a week, the TA also needs to notify the doctors to diagnose the patient within one day	bounded 3 cause 2 response; scope global	no
TA ($Prop_{21}$)	if the number of notifications to help a certain patient is less than five times during a month, the TA should also notify doctors to diagnose the patient to see whether the patient is fine	bounded 5 cause 2 response; scope global	yes
OJA ($Prop_{22}$)	if the credit of a user is verified to be valid, the user can eventually obtain a video within 2 minutes	bounded response; scope global	no
OJA ($Prop_{23}$)	if the user orders the email from VA, VA will periodically (every week) send a new video list to the user according to the user's watching habit	periodic recurrence; scope after	yes
OJA ($Prop_{24}$)	if there is no satisfied video in KB, VA will send the query request to other providers, then some providers respond to the request within 10 seconds. After VA provides a user with a video list (with respect to textual description of related service information) that is composed of all periodically collected responses, the user will read the video list and selects one of the providers for getting and watching the video within 20 seconds	bounded 2 cause 1 response and bounded 1 cause 2 response ; scope global	yes
OJA ($Prop_{25}$)	when watching a video, a user does not satisfy with the poor quality of the current video, she cannot watch the video for more than half a minute	maximum duration; scope global	no
TM ($Prop_{26}$)	if the user invokes the travel service (i.e. he/she sends the travel requirements to the TM service), the probability that the TM service responds to the user with the result of three service invocationsfindDuration, calculateTime and checkSchedule within 3.2 seconds	bounded 1 cause 3 response; scope global	yes
TM ($Prop_{27}$)	after the user orders the desired travel, the travel schedule will be sent to the user within 10 seconds	bounded 1 cause 4 response; scope global	yes

For Research Only

The analysis results are then shown to designers. They can further analyse and correct the possible errors in the system. In our example $Prop_1$, $Prop_2$ and $Prop_4$ are validated as correct, whereas $Prop_3$ and $Prop_5$ are false. From the error trace of $Prop_3$, we find that the users can obtain patient's diagnostic records from EMR without patient's permission. After adding a new activity [invoke]GetPermission to the corresponding patient whose medical data is used in the BPEL process, $Prop_3$ holds. From the error trace of $Prop_5$, we find that it is possible that within 30 seconds, the activity `[invoke]DeliveryLogisticsCompanies (detailMedicine)` does not happen. Consequently, the TA for $Prop_5$ goes to the accepting state and a violation against $Prop_5$ is observed.

In order to check probabilistic property, we systematically inject the correct implementations with a predefined failure rate(fr). To simulate a failure, a correct event such as 'confirmOK' is replaced by a 'confirmFail' or a 'confirmTimeout' message. In each experiment, we set statistical test parameters $\alpha = 0.1$, $\beta = 0.1$ and indifferent region as $H_0 - H_1 = 0.04$. We injected a failure that is 2% below or above the indifferent region. The validated results for properties $Prop_6$ and $Prop_7$ are shown in Fig. 16. For $Prop_6$, we injected 9% (in the indifference region), 7% (2 above H_0) and 15% (2 below H_1). For $Prop_7$, we injected 7% (in the indifference region), 3% (2 above H_0) and 11% (2 below H_1). From the curves, we can see that the test correctly accepts or rejects H_0 for these samples. However, the number of samples are different due to the different probabilities. For $Prop_6$, 100 samples are enough. But for $Prop_7$ 100 samples are not enough to draw the conclusion. For example, it needs 125 samples to accept H_0. When the probability is set at more than 98%, the tool cannot draw any conclusion within 200 samples. Furthermore, when the probabilities are in the indifferent region, it is very hard to draw any conclusion with our monitor.

5 Experimental validation

In previous sections, we have already shown how to use WS-PSC monitor to monitor temporal, timing and probabilistic properties in composite service with PSC/TPSC/PTPSC specifications. To demonstrate the generality of WS-PSC monitor, this section will first apply it to three other case studies and then conduct some experiments to analyse its performance. The tool has been developed on the Eclipse Rich Client Platform (RCP). It has 122 Java classes and about 20000 lines of codes [25].

5.1 Case studies

In this subsection, we will describe three case studies: TA (TelecommunicationAssistant), OJA (OntheJobAssistant) and TM (TravelManagement). These three case studies come from real industrial requirements and have already been widely used in web service research.

TelecommunicationAssistance [26, 27] composite service (TA) is a software- and telecommunication-based service that is designed to help patients who need daily assistance in remote areas. TA service interacts with four partner

© The Institution of Engineering and Technology 2013

Table 3 Informal descriptions and the corresponding patterns of the monitored probabilistic properties

Id	Informal descriptions	Patterns
TA ($Prop_{28}$)	after sending a patient's medical parameters to the Lab, TA receives a '*high*' result within one hour. TA will immediately notify the assistants nearest to the patient and must receive a response from them within one hour with 95% probability	probabilistic 2 cause 2 response
TA ($Prop_{29}$)	if for a certain patient the number of results with a 'high' criticality is more than three times during a week, the probability for TA to notify the doctors to diagnose the patient within one day is 90%	probabilistic 3 cause 2 response
OJA ($Prop_{30}$)	if the credit of a user is verified to be valid, the probability that the user can eventually obtain a video within 2 minutes is 90%	probabilistic response
OJA ($Prop_{31}$)	when watching a video, the probability that a user will cancel the video within half a min because she does not satisfy with the poor quality of the current video is 45%	maximum duration; scope probabilistic precedence
TM ($Prop_{32}$)	if the user invokes the travel service (i.e. he/she sends the travel requirements to the TM service), the probability that the TM service responds to the user with the result of three services invocationsfindDuration, calculateTime and checkSchedule within 3.2 seconds is 89%	probabilistic 1 cause 3 response
TM ($Prop_{33}$)	after the user orders the desired travel, the probability that the travel schedule will be sent to the user within 20 seconds is 85%	probabilistic constrained response

For Peer Review Only

services, including Patient, Medical Laboratory (Lab), Assistant and Doctor. Patient needs basic medical support in order to control their health. Medical Laboratory (Lab) can analyse the medical parameters of patients and replies to TA by sending the results. Assistant may be a patient's relative who will take over the role of a medical assistant. Doctor is in charge of medical decisions, such as changes of treatments or diagnosis.

OntheJobAssistant(OJA) [28] is a composite service to provide specialists with a variety of video help, consisting of four services: Knowledge Base (KB), Virtual Assistant (VA), Bank and Other Providers. KB can store various video information. VA is a medium between a user and KB, and submits a user's search request to KB. Bank is responsible for verifying whether the user's credit is valid. Other Providers: whenever video information required by the user is unavailable in KB, VA submits the user's search request to other providers.

Travel Management (TM) [29] is a composite service used to help users book a travel package online that suit their requirements. The system interacts with the following participants and services: User, DataBase (DB), Hotel Service (HS) and Traffic Service (TS). The DataBase stores various information about travel. The Hotel Service and Traffic Service provide the corresponding hotel and traffic service.

WS-PSC monitor is used to analyse 10 temporal properties, 10 timing properties and 6 probabilities properties selected from the three case studies.

Table 1 ($Prop_8$ to $Prop_{17}$) shows the informal descriptions, the corresponding specification patterns and the monitoring results of the temporal properties of the three case studies. To show the usability of our scenario-based specification most properties belong to chain patterns which are not easily represented by traditional temporal logic-based specifications. As a simple example, $Prop_{11}$ is a 5 Stimulus-2 Response pattern which is very hard to be represented by temporal logic formula.

The results indicate that $Prop_{10}$ and $Prop_{12}$ are false. To identify the cause of the violations, we review the interactions between all the BPEL processes. According to the violation trace for $Prop_{10}$, we find that after the message `[invoke]alarm(high)` happens more than three times, the message `[invoke]changeDiagnosis` has not been sent to the doctors. Consequently, the Analyzer will go to the accepting state and a violation against $Prop_{10}$ is observed. After analysing the original BPEL codes, we notice that the original TA process does not have a counter to record the number of message `[invoke]alarm(high)`. After we add a counter in the process, the problem is solved. According to the violation trace for $Prop_{12}$, we see that the credit check of a user is ok, but the money in his bank account is less than the price of the video. Consequently, the user still cannot get the video. This property is true occasionally, as the balance of the user's bank account is generated randomly. In order to solve this problem, we let the process generate at least $200 for user's bank account since the prices of all the video is less than $200.

Table 2 ($Prop_{18}$ to $Prop_{27}$) shows the informal descriptions, the corresponding specification patterns and the monitoring

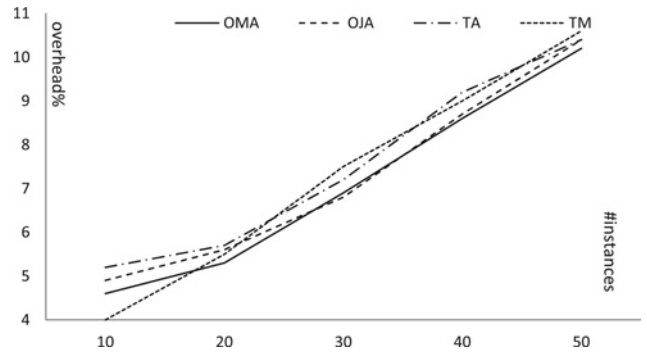


Fig. 17 *Performance characteristics of AOP-based interceptor of WS-PSC monitor*

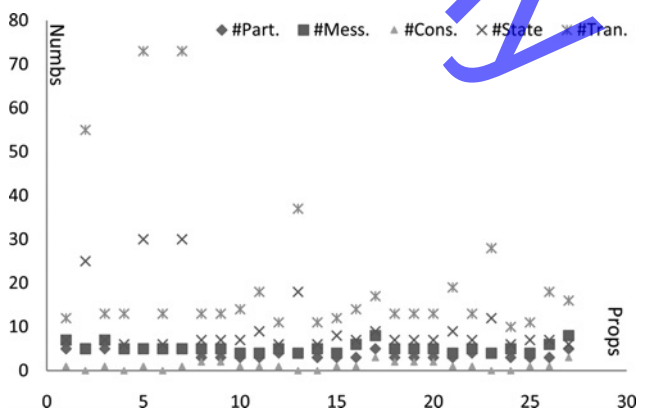


Fig. 18 *Monitored properties and the sizes of their generated automata*

© The Institution of Engineering and Technology 2013

doi: 10.1049/iet-sen.2012.0047

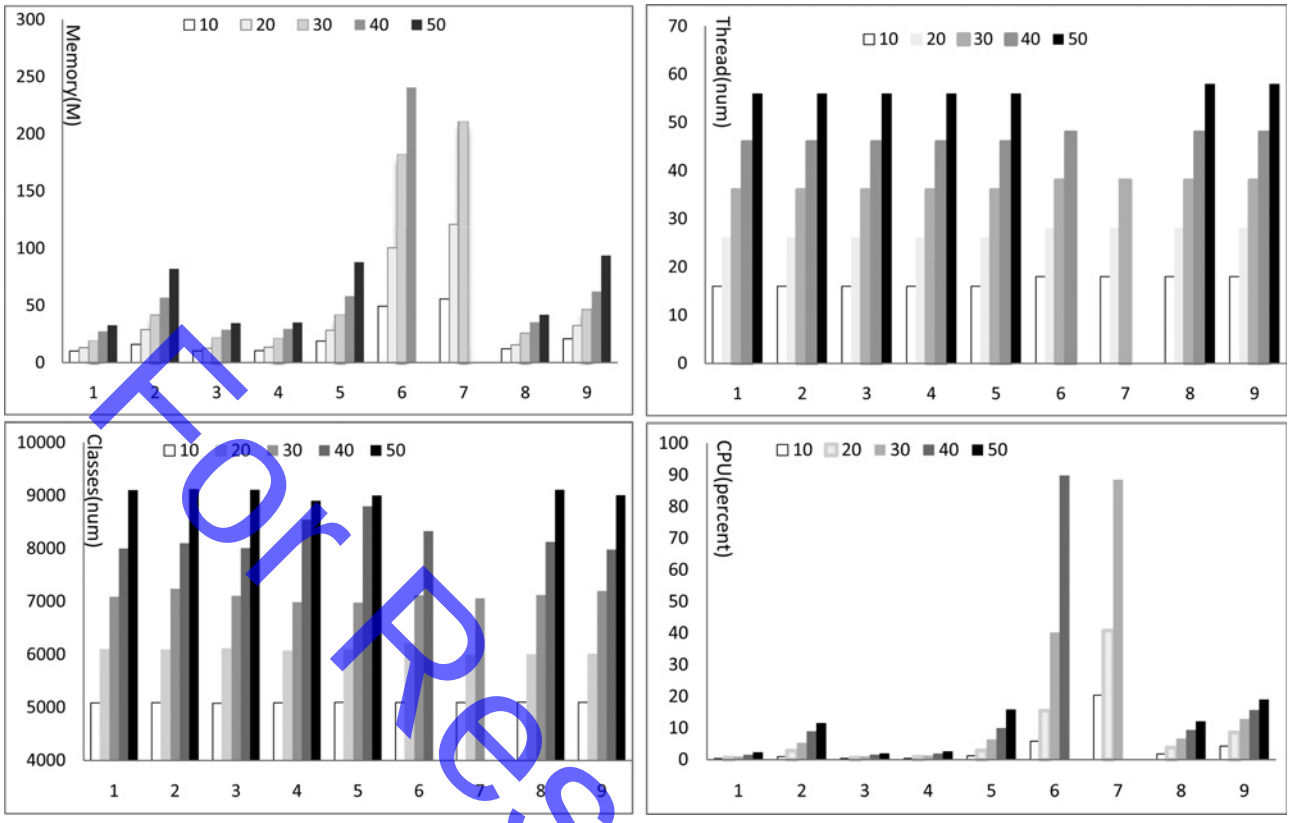


Fig. 19 *Validation and performance characteristics of WS-PSC monitor*

For Research Only

results of the timing properties of the three case studies. According to the results, $Prop_{20}$, $Prop_{22}$ and $Prop_{25}$ are false. $Prop_{20}$ and $Prop_{22}$ fail for the same reasons as those of $Prop_{10}$ and $Prop_{12}$, respectively. Compared with the temporal property, there is a new property violation. From the error trace, we find the problem is that some users want to cancel the video more than half a minute after they watch the video. These behaviours satisfy the temporal property $Prop_{15}$ but does not satisfy the timing property $Prop_{25}$.

Table 3 ($Prop_{28}$ to $Prop_{33}$) shows the informal descriptions and the corresponding specification patterns of the probabilistic properties of the three case studies. Normally, the SPRT process can correctly accept or reject H_0 for these properties. For the probability below 95%, 200 samples are enough to draw the conclusion. When the probability is more than 98%, 200 samples are not enough.

5.2 Performance analysis

In this subsection, we will do a critical performance analysis of the WS-PSC monitor tool. Two key factors that affect the performance of the tool are the AOP-based interceptor and the analyser engine. AOP-based interceptor will affect the execution time of the whole service since it will synchronise the execution of the service. As the analysers and the web service run concurrently, the execution time of the whole service should not be affected. As a platform for our experiments, we used a Windows-based PC equipped with a 1.83 GHz dual-core processor and 2 GB of main memory. The WS-BPEL engine (in the experiment we use Eclipse 3.4 with BPEL Designer plug in) ran on Tomcat 6.0 and Apache ODE.

Intercepting messages using AOP-based technology does not affect too much the performance of the system. We do some experiments to study the time overhead for using AOP-based intercepting. We first run the original four composite services with different instances and then run the four composite services with AOP and different instances. The results are shown in Fig. 17. From the figure, we can see that this overhead increases linearly with the number of instances. It take around 5% (4.6, 4.9, 5.2, 4.0%) overhead when the instances is 10. It will increase to around 10% (10.2, 10.4, 10.4, 10.6%) when the instances is 50. This is also in line with the results from other researchers; the overload for interceptor is within 5% [30] when the instances is small and it will increase to about 10% when the instances increase to about 50.

In the following, we will focus on investigating the performance of the Analyzers. There are two factors that affect their performance. One is the scale of the generated FA or TA, and the other is the number of service instances running in the engine. For the first factor, the detailed statistics for the monitored properties are illustrated in Fig. 18, where '#Part.' denotes the number of partners involved in the composite services; '#Mess.' is the number of messages exchanged between partners; '#Cons.' is the number of unwanted or wanted constraints in the PSC specifications. '#State' is the number of states in the generated automata and '#Tran.' is the number of transitions in the generated automata. For $Prop_{28}$ to $Prop_{33}$, they generate the same statistics as those in the experiment for timing properties, and are not shown in Fig. 18. Note that all of the constructed automata have fewer than 73 transitions. For the second factor, we run the service with different service instances.

© The Institution of Engineering and Technology 2013

For the Analyzers of temporal and timing properties, the composite service generates a large number of messages. However, our monitors receive just those within the PSC specifications. Furthermore, the intercepted events are never stored. Thus, we hope that the monitor does not produce a significant performance overhead. For the Analyzer of probabilistic properties, we have to record the run-time information to support SPRT process. Since there are 33 properties, we choose some properties as examples to analyse the performance, whereas the other properties can be analysed in a similar way and are omitted. In the experiments, we use Jconsole [http://java.sun.com/developer/technicalArticles/J2SE/jconsole.html] as a tool to record the heap memory usage, the number of Java classes, the number of collected threads and the CPU usage for the WS-PSC monitor.

The results are collected in Fig. 19. We do the experiments 10 times and record the average value. For the horizontal axis in each figure, numbers 1 to 7 mean the properties $Prop_1$ to $Prop_7$ whereas 8 and 9 mean the improved implementation of $Prop_6$ and $Prop_7$ called $Prop'_6$ and $Prop'_7$. Each figure shows the results for five sets of service instances of (10, 20, 30, 40, 50).

When the number of service instances is 10, the consumed memory is about 10.1 to 18.8 M for $Prop_1$ to $Prop_5$; the collected threads and Java classes are almost the same. The consumed CPU usage is from 0.35 to 1.2%. Consequently, the first factor has minimal effect on the performance since the transitions for $Prop_1$ is 12 and for $Prop_5$ is 73 according to Fig. 18. Furthermore, the additional checking of timing information also has minimal effect on the performance. However, when the number of service instances is increased to 50, the consumed memory is 32.6 M for $Prop_1$ and 87.8 M for $Prop_5$; the collected threads and Java classes are also almost the same; the consumed CPU usage is from 2.34 to 15.86%. From the results we can see that the consumed memory, threads, classes and CPU will increase when the number of service instance increases.

For probabilistic properties, the results are shown under number 6 and 7. We can see that the threads and classes are slightly increased due to the running of SPRT process. We first collect all the generated run-time messages, then we do the SPRT process for every 5000 messages. Consequently, the consumed memory and CPU (55.6 M and 20.4% for $Prop_7$) are dramatically increased due to storing a great number of messages. When the number of service instances is 40, the memory and CPU increase to 240.5 M and 89.8%, respectively, for $Prop_6$ and ran out of CPU for $Prop_7$. When the number of service instances is 50, we cannot monitor $Prop_6$ and $Prop_7$ neither.

To resolve this problem, we just store the TA-based checking results. From the results in number 8 and 9, we can see that the tool can monitor $Prop_6$ and $Prop_7$ since the consumed memory and CPU (93.6 MB and 19% for $Prop_7'$) have been dramatically reduced.

5.3 Discussions

The four case studies clearly demonstrate the intuitiveness and expressiveness of our graphical specification PSC, TPSC and PTPSC (i.e. we can use PSC/TPSC/PTPSC to represent 33 properties). Furthermore, WS-PSC monitor can monitor all 33 properties at run-time with reasonable overhead. For temporal and timing properties, the tool has a reasonable overhead when the number of service instances increases from 10 to 50. For probabilistic properties the tool requires more processing and memories, since it needs to record previous results. With an improved implementation, the tool can still monitor these probabilistic properties when the number of service instances is within 50. However, the proposed tool suffers from the following limitations.

5.3.1 Full automation: Although PSC, TPSC and PTPSC have graphical interfaces, they are still somewhat difficult to write and prone to mistakes when the number of messages grows. Consequently, to reduce this limitation, it is important to develop an approach to help designers automatically derive PSC, TPSC and PTPSC specifications from textual requirements in the future.

5.3.2 Scalability: Although the four case studies are from real industrial requirements, they are devised and invented in academic setting. Furthermore, they are small and not complex. Consequently, the scalability of the approach needs to be tested with larger case studies. As already shown in our experiments, the memory, CPU usage, and the time consumed for the monitoring process will increase with the number of properties and the number of the monitored messages; especially for probabilistic monitors since this kind of monitors need to record some historical information. On the one hand, new and efficient monitoring algorithms can be developed to reduce this limitation. On the other hand, the use of multi-core platforms [31, 32] can also reduce this limitation because it can make the monitoring cost acceptable.

5.3.3 Precision: For probabilistic properties, if the monitored probability is inside the indifference region between H_0 and H_1 the monitoring result is not accurate. Another specific limitation is the ability to monitor probabilistic properties with extreme probabilities (e.g. when the probability is close to 1 or 0). The number of required monitoring samples runs increase dramatically in order to draw a conclusion. New approaches are needed to solve these problems.

6 Related work

In recent years, a large number of approaches have been proposed for monitoring web service compositions. This section will provide a survey of existing tool support for monitoring of behaviour of composite services against various properties.

Baresi *et al.* [26] propose a framework called dynamo-AOP for monitoring functional and non-functional properties that a service provider should fulfil. Its architecture is based on the dynamic aspectisation of the BPEL engine executing the monitored service compositions, achieved by using AspectJ as an AOP language. They consider an existing monitoring approach based on ALBERT, which is a temporal logic language suitable for asserting both functional and non-functional properties, and show how to obtain the efficient run-time verification of ALBERT formulae.

Manageable and adaptive service composition (MASC) is a policy-based middleware for monitoring and control of composite web services execution [33]. WS-Policy4MASC language extends WS-Policy by defining new types of monitoring and control policy assertions. It is used to specify monitoring properties related to business exceptions and run-time failures. The approach provides synchronous and asynchronous monitoring at both the SOAP messaging layer and the process orchestration layer, greater diversity of monitoring and control constructs, as well as the

For Review Only

© The Institution of Engineering and Technology 2013

doi: 10.1049/iet-sen.2012.0047

IET Softw., 2013, Vol. 7, Iss. 4, pp. 222–248
doi: 10.1049/iet-sen.2012.0047

Table 4 Comparison of existing run-time monitoring approaches for web service composition with tool support

Name	Collaboration type	Property specification	Property type	Timeliness	Validation technique	Tool type	Performance measurement
Dynamo [26]	WS-BPEL	logic-based/WSCoL and Timed WSCoL	pre-, post condition, temporal and timing properties	synchronous and asynchronous	assertion checking and timed automata based trace analysing	available/ DynamoAOP	yes
Erradi *et al.* [33]	WS-BPEL	XML-based/WS-Policy4 MASC	functional and non-functional properties	synchronous and asynchronous	policy checking	internal/MASC Middleware	yes
Mahbub and Spanoudakis [34]	BPEL4WS	process-based/event calculus	QoS properties	asynchronous	condition evaluation service	Internal	no
SLAngMon [35]	interaction between a client and the provider	XML-based/SLAng	non-functional properties	asynchronous	timed automata-based technique	available/ SLAngMon	yes
Simmonds *et al.* [29, 36]	WS-BPEL	scenario-based/UML 2.0 Sequence Diagrams	safety and bounded liveness properties	asynchronous	automata-based trace analysing	internal/a tool in IBM WebSphere	no
Wang *et al.* [30]	interaction between clients and the provider	XML-based/WSCDL	assertion, functional and non-functional properties	synchronous and asynchronous	finite state automata-based checking	internal	yes
Cremona [39]	interaction between a client and the provider	XML-based/WSLA	functional and non-functional properties	synchronous	implementation specific techniques	industrial/ Cremona	no
Colombo [40]	BPEL4WS	XML-based/WS-policy	non-functional properties	asynchronous or post-mortems	manually	industrial/ Colombo	no
Hallé and Villemaire [37]	interaction between a client and the provider	LTL-FO+	temporal properties with data	synchronous	on-the-fly algorithm	internal	yes
Pesic *et al.* [38]	workflow management systems	DECLARE	temporal properties with parameters	synchronous	process mining	available	yes
WS-PSC monitor	WS-BPEL	PSC/TPSC/PTPSC	temporal, timing and probabilistic properties	asynchronous	automata and TA-based analysing and hypothesis testing	internal/WS-PSC monitor	yes

For Research Only

© The Institution of Engineering and Technology 2013

www.ietdl.org

externalisation of monitoring and adaptation actions from definitions of business processes. A prototype is implemented and evaluated on monitoring and adaptation scenarios from a stock trading case study. The performance studies indicate that MASC's overhead and scalability are acceptable.

Mahbub and Spanoudakis [34] proposed a framework for the run-time verification of requirements of service-based software systems. System events are collected at run time and stored in an event database. The properties are checked by means of an algorithm based on integrity constraint checking in temporal deductive databases. A prototype tool has been developed to demonstrate and evaluate the framework. The approach has been applied in the car rental system service composition. However, no performance measures have been reported.

Raimondi *et al.* [35] show how timeliness constraints, such as latency, throughput, availability and reliability, in formal service-level agreements can be translated into TA. They attach time stamps to SOAP messages and consider these messages as timed letters. They are then able to reduce the problem of detecting SLA violations to acceptance of timed words by the TA that have been derived from the SLAs.

Simmonds *et al.* [36] proposed to use run-time monitoring of conversations between partners as a means of checking behavioural correctness of web services. They identified a subset of UML 2.0 SD as a property specification language and showed that it is sufficiently expressive for capturing safety and liveness properties. By transforming these diagrams to automata, this approach can check finite execution traces against the specification. They showed how the language can be used to specify the specification property system. They described an implementation of the approach as part of an industrial system. Finally, they discussed the experience of specifying and monitoring a number of properties from three existing applications.

Wang *et al.* [30] proposed an online monitoring approach for web service requirements. It includes a pattern-based specification of service constraints that correspond to service requirements. The monitoring framework uses different probes and agents to collect events and data that are sensitive to requirements. The framework analyses the collected information against the pre-specified constraints, so as to evaluate the behaviour and use of web services. The prototype implementation and the corresponding experiments with a case study show that the approach is effective and flexible, and the monitoring cost is reasonable.

In [37], the authors present an algorithm for the monitoring of run-time message contracts with data. Their properties are expressed in LTL-FO+, an extension of linear temporal logic that allows first-order quantification over the data inside a trace of XML messages. An implementation of this algorithm can transparently enforce an LTL-FO+ specification using Java applet. Violations of the specification are reported on-the-fly and it can prevent erroneous or out-of-sequence XML messages from being exchanged. Experiments on commercial case study indicate that LTL-FO+ is an appropriate language for expressing their message contracts, and that its processing overhead on sample traces is acceptable both for client-side and server-side enforcement architectures.

Pesic *et al.* [38] propose DECLARE, a prototype of a workflow management system that uses a constraint-based process modelling language for the development of declarative models describing loosely-structured processes. They show how DECLARE can support loosely-structured processes without sacrificing important WFMSs features like user support, model verification, analysis of past executions using process mining techniques and changing models at run-time.

Cremona [39] is a tool from IBM devised to help clients and providers in the negotiation and life-cycle management of WS-agreements. It provides 'Status Monitor' component, which helps in deciding whether a negotiation proposal should be accepted or refused, on the basis of system available resources and the terms of an agreement. Once an agreement has been accepted by the client and the provider, its validity is checked at run-time by a 'Compliance Monitor', which can detect violations, predict violations to be occurred and take corrective actions.

Colombo [40] provide a lightweight middleware for service-oriented architectures that support BPEL. It can support declarative service descriptions, such as WS-policy. It can intercept messages before they leave the system or before they are processed, and can use a pipe of dedicated policy-specific verifiers to validate messages with respect to a certain policy.

Comparison of existing run-time monitoring approaches is illustrated in Table 4 according to different parameters. These comparison parameters follow the taxonomy in [3] with some modifications or extensions. 'Collaboration type' denotes the composition type of services. Our tool follows most of the common approaches and chooses BPEL-based web service composition. 'Property specification' indicates the type of property specification used to specify properties. Our tool uses scenario-based notations to represent desired properties in event-based web service composition and provides users with a completely graphical front-end. 'Property type' indicates the kind of properties specified by the language (temporal, timing or probabilistic). Our tool can monitor three kinds of properties. 'Validation technique' shows the adopted technique used for validation at run-time. Our tool also follows most common approaches and uses automata and time automata-based validation technique. 'Timeliness' indicates that the monitoring activity is performed (post-mortem, synchronous or asynchronous). In order to detect errors as early as possible and reduce the performance overhead, our tool chooses asynchronous style. 'Tool type' means the approach is supported by a tool (industrial or internal (available) prototype). 'Performance measurement' means whether the performance of the tool is measured or not. To demonstrate the generality of our proposed approach, the performance of our tool has been studied based on four case studies.

7 Conclusions and future work

This paper demonstrates the use of WS-PSC tool chain to monitor temporal, timing and probabilistic properties in BPEL-based composite services using existing graphical specification formalisms. Compared to other approaches, our approach provides a completely graphical front-end for software designers so that they do not have to deal with any particular textual and logical formalisms. Furthermore, the four case studies show that our tool can monitor three kinds of properties with reasonable performance. Finally, the comparative performance of different properties and service instances show some important results: (i) The interceptor does not affect too much the performance of the system; (ii) the scale of the generated monitor of analyser has minimal effect on the performance since the number of transitions

For Review Only

© The Institution of Engineering and Technology 2013

doi: 10.1049/iet-sen.2012.0047

for normal properties is within 100; (iii) the checking of temporal and timing properties gives almost the same performance, whereas checking of probabilistic properties requires more CPU and memories since it needs to record previous results; (iv) the consumed memory and CPU will increase with the number of service instances.

Several directions for future work are possible. First, to automate the elicitation of properties represented by PSC/TPSC/PTPSC specifications could be an interesting future research task. Initial work has already been done by Autili and Pelliccione [41] to automatically derive PSC specifications from textual requirements. Further work is to derive TPSC and PTPSC specifications from textual requirements using the similar idea. Secondly, our monitoring approach detects errors too late, after the failure arises. In the future we plan to define advanced monitors with the ability to predict and prevent the potential errors. The proposed new approach can 'look ahead' in the near execution future, and predict potential sources of failures. Thirdly, our statistical approach is not accurate if the monitored probability is inside the indifference region between H_0 and H_1. In order to deal with this problem, we plan to use Bayesian sequential hypothesis testing technique [42] since it has already shown faster verification ability than state-of-the-art techniques in statistical model checking.

8 Acknowledgements

This work is supported by the National Natural Science Foundation of China (No. 91118007, No. 61021062, No. 61202097, No. 61202136) and by the National 863 High-Tech Program of China (No. 2011AA010103, No. 2012AA011205), China Postdoctoral Science Foundation (Grant No. 2012T50489 and No. 2011M500897), and Specialized Research Fund for the Doctoral Program of Higher Education (No. 20120094120009).

9 References

1 WS-BPEL: 'Web services business process execution language version 2.0, committee specification' (OASIS, 2007)
2 Leucker, M., Schallhart, C.: 'A brief account of runtime verification', *J. Log. Algebr. Program.*, 2009, **78**, (5), pp. 293–303
3 Delgado, N., Gates, A.Q., Roach, S.: 'A taxonomy and catalog of runtime software-fault monitoring tools', *IEEE Trans. Softw. Eng*, 2004, **30**, (12), pp. 859–872
4 Zhang, P., Li, W., Wan, D., Grunske, L.: 'Monitoring of probabilistic timed property sequence charts', *Softw.: Pract. Experience*, 2011, **41**, (7), pp. 841–866
5 Holzmann, G.J.: 'The logic of bugs'. Proc. Tenth ACM SIGSOFT Symp. on the Foundations of Software Engineering (FSE-02), Vol. 27, 6 of Software Engineering Notes, New York, 2002, pp. 81–88
6 Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: 'Property specification patterns for finite-state verification'. Proc. 21st Int. Conf. Software Engineering (ICSE99), 1999, pp. 411–420
7 ITU: 'Message Sequence Chart (MSC): International Telecomm' (Union, 1999)
8 OMG: 'UML 2.1.1 superstructure' (Object Management Group (OMG), 2006)
9 Damm, W., Harel, D.: 'LSCs: breathing life into message sequence charts', *Formal Meth. Syst. Des.*, 2001, **19**, (1), pp. 45–80
10 Autili, M., Inverardi, P., Pelliccione, P.: 'Graphical scenarios for specifying temporal properties: an automated approach', *Autom. Softw. Eng.*, 2007, **14**, (3), pp. 293–340
11 Zhang, P., Li, B., Grunske, L.: 'Timed property sequence chart', *J. Syst. Softw.*, 2010, **83**, (3), pp. 371–390
12 Zhang, P., Grunske, L., Tang, A., Li, B.: 'A formal syntax for probabilistic timed property sequence charts'. Proc. ASE, 2009, pp. 500–504
13 Konrad, S., Cheng, B.H.C.: 'Real-time specification patterns'. 27th Int. Conf. Software Engineering (ICSE 05), 2005, pp. 372–381
14 Grunske, L.: 'Specification patterns for probabilistic quality properties'. 30th Int. Conf. Software Engineering (ICSE 2008), Leipzig, Germany, 10–18 May 2008, pp. 31–40
15 Gerth, R., Peled, D., Vardi, M.Y., Wolper, P.: 'Simple on-the-fly automatic verification of linear temporal logic'. PSTV, 1995, pp. 3–18
16 Alur, R., Courcoubetis, C., Dill, D.: 'Model-checking in dense real-time', *Inf. Comput.*, 1993, **104**, (1), pp. 2–34
17 Alur, R., Dill, D.L.: 'A theory of timed automata', *Theor. Comput. Sci.*, 1994, **126**, (2), pp. 183–235
18 Wald, A.: 'Sequential tests of statistical hypotheses', *Ann Math. Stat.*, 1945, **16**, (2), pp. 117–186
19 Aho, A., Sethi, R., Ullman, J.: 'Complilers: principles, techniques and tools' (Addison-Wesley, 1986)
20 Kiczales, G., Lamping, J., Mendhekar, A., *et al.*: 'Aspect-oriented programming'. ECOOP, 1997, pp. 220–242
21 Bauer, A., Leucker, M., Schallhart, C.: 'Monitoring of real-time properties'. FSTTCS 2006: 26th Int. Conf. Foundations of Software Technology and Theoretical Computer Science, 2006, pp. 260–272
22 Bauer, A., Leucker, M., Schallhart, C.: 'Runtime verification for LTL and TLTL', *ACM Trans. Softw. Eng. Methodol.*, 2011, **20**, (4), p. 14
23 Zhang, P., Li, B., Muccini, H., Sun, M.: 'An approach to monitor scenario-based temporal properties in web service compositions'. DeWeb08 in conjunction with APWeb08, Volume 4977 of LNCS, 2008, pp. 144–154
24 Zhang, P., Li, B., Sun, M.: 'Extending PSC for monitoring the timed properties in composite services'. Proc. APSEC, 2008, pp. 335–342
25 Zhang, P., Su, Z., Zhu, Y., Li, W., Li, B.: 'WS-PSC monitor: a tool chain for monitoring temporal and timing properties in composite service based on property sequence chart'. First Int. Conf. Runtime Verification (RV 2010), LNCS, 2010, pp. 485–489
26 Baresi, L., Bianculli, D., Ghezzi, C., Guinea, S., Spoletini, P.: 'A timed extension of WSCoL'. IEEE Int. Conf. Web Services, 2007, pp. 663–670
27 Calinescu, R., Grunske, L., Kwiatkowska, M.Z., Mirandola, R., Tamburrelli, G.: 'Dynamic QoS management and optimization in service-based systems', *IEEE Trans. Softw. Eng.*, 2011, **37**, (3), pp. 387–409
28 Li, W., Wang, Z.: 'Monitoring composite services with universal modal sequence diagrams'. APSEC, 2009, pp. 69–76
29 Simmonds, J., Gan, Y., Chechik, M., *et al.*: 'Runtime monitoring of web service conversations', *IEEE Trans. Serv. Comput.*, 2009, **2**, (3), pp. 223–244
30 Wang, Q., Shao, J., Deng, F., *et al.*: 'An online monitoring approach for web service requirements', *IEEE Trans. Serv. Comput.*, 2009, **2**, (4), pp. 338–351
31 Nagarajan, V., Gupta, R.: 'Runtime monitoring on multicores via oases', *Oper. Syst. Rev.*, 2009, **43**, (2), pp. 15–24
32 Yang, L., Yu, L., Tang, J., Wang, L., Zhao, J., Li, X.: 'McC++ /Java: enabling multi-core based monitoring and fault tolerance in C++/ Java'. ICECCS, 2010, pp. 255–256
33 Erradi, A., Maheshwari, P., Tosic, V.: 'WS-policy based monitoring of composite web services'. ECOWS, 2007, pp. 99–108
34 Mahbub, K., Spanoudakis, G.: 'Monitoring WS-agreements: an event calculus-based approach', in Baresi, L., Nitto, E.D. (Eds.): 'Test and analysis of web services' (Springer, 2007), pp. 265–306
35 Raimondi, F., Skene, J., Emmerich, W.: 'Efficient online monitoring of web-service SLAS'. SIGSOFT FSE, 2008, pp. 170–180
36 Simmonds, J., Chechik, M., Nejati, S., Litani, E., O'Farrell, B.: 'Property patterns for runtime monitoring of web service conversations'. Runtime Verification RV'08, Volume 5289 of LNCS, 2008, pp. 137–157
37 Hallé, S., Villemaire, R.: 'Runtime enforcement of web service message contracts with data', *IEEE Trans. Serv. Comput.*, 2012, **5**, (2), pp. 192–206
38 Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: 'DECLARE: full support for loosely-structured processes'. 11th IEEE Int. Enterprise Distributed Object Computing Conf. (EDOC 2007), 15–19 October 2007, Annapolis, Maryland, USA, 2007, pp. 287–300
39 Ludwig, H., Dan, A., Kearney, R.: 'Cremona: an architecture and library for creation and monitoring of WS-Agreements'. ICSOC, 2004, pp. 65–74
40 Curbera, F., Duftler, M.J., Khalaf, R., Nagy, W., Mukhi, N., Weerawarana, S.: 'Colombo: Lightweight middleware for service-oriented computing', *IBM Syst. J.*, 2005, **44**, (4), pp. 799–820
41 Autili, M., Pelliccione, P.: 'Towards a graphical tool for refining user to system requirements', *Electr. Notes Theor. Comput. Sci.*, 2008, **211**, pp. 147–157
42 Jha, S.K., Clarke, E.M., Langmead, C.J., Legay, A., Platzer, A., Zuliani, P.: 'A Bayesian approach to model checking biological systems'. CMSB, 2009, pp. 218–234

For Research Only

© The Institution of Engineering and Technology 2013

9 Appendix: The formal semantics of PSC

9.1 Formal denotational semantics of PSC

Before giving the detailed semantics, we first give some explanation of basic concepts and notations used in the semantics. The denotational semantics of an **arrowMsg** of PSC with constraint can be defined as: $[[(<t, a, C, C', p, f>, op)]]^{trace}$, where $t \in \{e:, r:, f:\}$ is the type of message. a is the message exchanged by sending service C and receiving service C'. $l = C.a.C'$ is the message label. $p, f \in \{\lambda, \bullet b, =>g, \neq>g\}$, p means past constraint, f means future constraint, λ means empty constraint, $\bullet b$ means unwanted message constraint, $\Rightarrow g$ means wanted chain constraint and $\neq>g$ means unwanted chain constraint. $b = \{m_1, m_2, \ldots, m_n\}$ is the unwanted message set and $g = (m_1, m_2, \ldots, m_n)$ is the wanted or unwanted message sequence. $op \in \{\text{nop}, \text{strict}\}$, where nop means loose order whereas strict means strict order. $trace \in \{VC, UDC, IVT\}$, where VC means the set of validated continuations, UDC the set of undecided continuations and IVT the set of invalidated traces, since there is no continuation any more. L means the message set exchanged by different services, and $\alpha \in L$, α^* means finite traces whereas α^∞ means infinite traces. Since we cannot monitor infinite traces, in practice, when the bound of a trace reach a particular large number, we view it as infinite trace. $\alpha^{*/\infty}$ means the trace can be finite and infinite. In the following, we define the denotational semantics of PSC.

9.1.1 Basic semantics: A.1.1–A.1.9 relate to the denotational semantics of Regular messages. Note that regular message does not have invalidated trace set since it does not need to happen.

A.1.1 *Regular* message without constraint:

$$[[(<e:, a, C, C', \lambda, \lambda>, \text{nop})]]^{VC} = \{\alpha \cdot l | \alpha \in (L\backslash\{l\})^*\}$$

$$[[(<e:, a, C, C', \lambda, \lambda>, \text{nop})]]^{UDC} = \{\alpha | \alpha \in (L\backslash\{l\})^{*|\infty}\}$$

A.1.2 *Regular* message with strict operator:

$$[[(<e:, a, C, C', \lambda, \lambda>, \text{strict})]]^{VC} = \{l\}$$

$$[[(<e:, a, C, C', \lambda, \lambda>, \text{strict})]]^{UDC} = \{\alpha | \alpha \in (L\backslash\{l\})^{*|\infty}\}$$

A.1.3 *Regular* message with past unwanted constraint:

$$[[(<e:, a, C, C', \bullet b, \lambda>, \text{nop})]]^{VC} = \{\alpha \cdot l | \alpha \in (L\backslash b)^*\}$$

$$[[(<e:, a, C, C', \bullet b, \lambda>, \text{nop})]]^{UDC} = \{\alpha | \alpha \in ((L\backslash b) \bigcap (L\backslash l))^{*|\infty}\}$$

A.1.4 *Regular* message with future unwanted constraint:

$$[[(<e:, a, C, C', \lambda, \bullet b>, \text{nop})]]^{VC} = \{\alpha \cdot l \cdot \beta | \alpha \in (L\backslash\{l\})^* \wedge \beta \in (L\backslash b)^*\}$$

$$[[(<e:, a, C, C', \lambda, \bullet b>, \text{nop})]]^{UDC} = \{\alpha\} \bigcup \{\beta \cdot l \cdot \gamma\}$$
$$\text{where} \quad (\alpha \in (L\backslash\{l\})^\infty) \wedge (\beta \in (L\backslash\{l\})^*) \wedge (\gamma \in b^{*|\infty})$$

The set of traces is an abbreviation of the set of traces $\{\alpha | \alpha \in (L\backslash\{l\})^\infty\} \bigcup \{\beta \cdot l \cdot \gamma | \beta \in (L\backslash\{l\})^* \wedge \gamma \in b^{*|\infty}\}$. We will use this kind of abbreviation in the rest of the Appendix.

A.1.5 *Regular* message with strict operator and future unwanted message constraint:

$$[[(<e:, a, C, C', \lambda, b>, \text{strict})]]^{VC} = \{l \cdot \beta | \beta \in (L\backslash b)^*\}$$

$$[[(<e:, a, C, C', \lambda, b>, \text{strict})]]^{UDC} = \{\alpha\} \bigcup \{l \cdot \gamma\}$$
$$\text{where} \ (\alpha \in (L\backslash\{l\})^{*|\infty}) \wedge (\gamma \in b^{*|\infty})$$

A.1.6 *Regular* message with past unwanted chain constraint: (see equation at the bottom of the page)

$$[[(<e:, a, C, C', \neq>, \lambda>, \text{nop})]]^{UDC} =$$
$$\{\beta_1\} \bigcup \{\beta_1' \cdot m_1 \cdot \beta_2\} \bigcup \cdots \bigcup \{\beta_1' \cdot m_1 \cdot \beta_2' \cdot \ldots \cdot \beta_n' \cdot m_n \cdot \gamma\}$$
$$\text{where} \ \left(\beta_i' \in (L\backslash\{m_i\})^* \wedge \beta_i \in (L\backslash\{m_i\})^{*|\infty} \wedge 1 \le i \le n\right) \wedge$$
$$(\gamma \in (L\backslash\{a\}^{*|\infty}))$$

A.1.7 *Regular* message with past wanted chain constraint:

$$[[(<e:, a, C, C', =>, \lambda>, \text{nop})]]^{VC} = \{\beta_1 \cdot m_1 \cdot \beta_2 \cdot m_2 \cdot \cdots \cdot \beta_n \cdot m_n \cdot \gamma \cdot l | \beta_i \in (L\backslash\{m_i\})^* \wedge 1 \le i \le n \wedge \gamma \in (L\backslash\{a\}^*)$$

$$[[(<e:, a, C, C', =>, \lambda>, \text{nop})]]^{UDC} = \{\beta_1\} \bigcup \{\beta_1' \cdot m_1 \cdot \beta_2\} \bigcup \cdots \bigcup \{\beta_1' \cdot m_1 \cdot \beta_2' \cdot \cdots \cdot \beta_n'\} \bigcup \{\beta_1' \cdot m_1 \cdot \beta_2' \cdot \cdots \cdot \beta_n' \cdot m_n \cdot \gamma\}$$
$$\text{where} \ \left(\beta_i' \in (L\backslash\{m_i\})^* \wedge \beta_i \in (L\backslash\{m_i\})^{*|\infty} \wedge 1 \le i \le n\right) \wedge$$
$$(\gamma \in (L\backslash\{a\}^{*|\infty})$$

A.1.8 *Regular* message with future unwanted chain

$$[[(<e:, a, C, C', \neq>, \lambda>, \text{nop})]]^{VC}$$
$$= \{\beta_1 \cdot l\} \bigcup \{\beta_1 \cdot m_1 \cdot \beta_2 \cdot l\} \bigcup \cdots \bigcup \{\beta_1 \cdot m_1 \cdot \beta_2 \cdot m_2 \cdot \ldots \cdot \beta_{n-1} \cdot m_{n-1} \cdot \gamma \cdot l\}$$
$$\text{where} \quad (\beta_i \in (L\backslash\{m_i\})^* \wedge 1 \le i \le n-1) \wedge (\gamma \in (L\backslash\{l\}^*))$$

For Research Only

© The Institution of Engineering and Technology 2013

doi: 10.1049/iet-sen.2012.0047

constraint:

$$[[(<e:, a, C, C', \lambda, \neq>>, \text{nop})]]^{VC}$$
$$= \{\alpha \cdot l\} \bigcup \{\alpha \cdot l \cdot \beta_1 \cdot m_1\} \bigcup \{\alpha \cdot m_1 \cdot \beta_2 \cdot L_2\}$$
$$\bigcup \cdots \bigcup \{\alpha \cdot l \cdot \beta_1 \cdot m_1 \cdot \beta_2 \cdot m_2 \cdot \cdots \cdot \beta_{n-1} \cdot m_{n-1}\}$$

where $(\alpha \in (L\backslash\{l\})^*) \wedge (\beta_i \in (L\backslash\{m_i\})^* \wedge 1 \le i \le n)$

$$[[(<e:, a, C, C', \lambda, \neq>>, \text{nop})]]^{UDC}$$
$$= \{\alpha\} \bigcup \{\alpha' \cdot \beta_1 \cdot \beta_2 \cdot \cdots \cdot \beta_n \cdot m_n \cdot l \cdot \gamma\}$$

where $(\alpha \in (L\backslash\{l\})^{*|\infty}) \wedge (\alpha' \in (L\backslash\{l\})^*) \wedge$
$(\beta_i \in (L\backslash\{m_i\})^* \wedge 1 \le i \le n) \wedge (\gamma \in L^{*|\infty})$

A.1.9 *Regular* message with future wanted chain constraint:

$$[[(<e:, a, C, C', \lambda, =>>, \text{nop})]]^{VC} =$$
$$\{\alpha \cdot l \cdot \beta_1 \cdot m_1 \cdot \beta_1 \cdot m_2 \cdot \cdots \cdot \beta_n \cdot m_n | \alpha \in (L\backslash\{l\})^*$$
$$\wedge \beta_i \in (L\backslash\{m_i\})^* \wedge 1 \le i \le n$$

$$[[<e:, a, C, C', \lambda, =>>, \text{nop})]]^{UDC} = \{\alpha\} \bigcup \{\alpha' \cdot l \cdot \beta_1\}$$
$$\bigcup \{\alpha' \cdot l \cdot \beta_{1'} \cdot m_1 \cdot \beta_2\} \bigcup \cdots \bigcup \{\alpha' \cdot l \cdot \beta_{1'} \cdot m_1 \cdot \beta_{2'} \cdot \cdots \cdot \beta_{n-2}\}$$
$$\bigcup \{\alpha' \cdot l \cdot \beta_{1'} \cdot m_1 \cdot \beta_{2'} \cdot \cdots \cdot \beta_{n-1} \cdot m_{n-1}\}$$

where $(\alpha \in (L\backslash\{l\})^{*|\infty}) \wedge (\alpha' \in (L\backslash\{l\})^*) \wedge$
$(\beta_i \in (L\backslash\{m_i\})^{*|\infty} \wedge \beta_i' \in (L\backslash\{m_i\})^* \wedge 1 \le i \le n-1)$

A.1.10–A.1.18 relate to the denotational semantics of Required messages.

A.1.10 *Required* message without constraint:

$$[[(<r:, a, C, C', \lambda, \lambda>, \text{nop})]]^{VC} = \{\alpha \cdot l | \alpha \in (L\backslash\{l\})^*\}$$
$$[[(<r:, a, C, C', \lambda, \lambda>, \text{nop})]]^{UDC} = \{\alpha | \alpha \in (L\backslash\{l\})^*\}$$
$$[[(<r:, a, C, C', \lambda, \lambda>, \text{nop})]]^{IVT} = \{\alpha | \alpha \in (L\backslash\{l\})^\infty\}$$

A.1.11 *Required* message with strict operator:

$$[[(<r:, a, C, C', \lambda, \lambda>, \text{strict})]]^{VC} = \{l\}$$
$$[[(<r:, a, C, C', \lambda, \lambda>, \text{strict})]]^{UDC} = \{\varepsilon\}$$
$$[[(<r:, a, C, C', \lambda, \lambda>, \text{strict})]]^{IVT} = \{\alpha | \alpha \in (L\backslash\{l\})^*\}$$

A.1.12 *Required* message with past unwanted constraint:

$$[[(<r:, a, C, C', \bullet b, \lambda>, \text{nop})]]^{VC} = \{\alpha \cdot l | \alpha \in (L\backslash b)^*\}$$
$$[[(<r:, a, C, C', \bullet b, \lambda>, \text{nop})]]^{UDC}$$
$$= \left\{\alpha | \alpha \in ((L\backslash b) \bigcap (L\backslash a))^*\right\}$$
$$[[(<r:, a, C, C', \bullet b, \lambda>, \text{nop})]]^{IVT} = \{\beta\} \bigcup \{\alpha \cdot \gamma\} \quad \text{where}$$
$$(\beta \in b^\infty) \wedge (\alpha \in (L\backslash b)^* \wedge \gamma \in (L\backslash a)^\infty)$$

A.1.13 *Required* message with future unwanted constraint:

$$[[(<r:, a, C, C', \lambda, \bullet b>, \text{nop})]]^{VC}$$
$$= \{\alpha \cdot l \cdot \beta | \alpha \in (L\backslash\{l\})^* \wedge \beta \in (L\backslash b)^*\}$$
$$[[(<r:, a, C, C', \lambda, \bullet b>, \text{nop})]]^{UDC} = \{\alpha | \alpha \in (L\backslash b)^*\}$$
$$[[(<r:, a, C, C', \lambda, \bullet b>, \text{nop})]]^{IVT}$$
$$= \{\alpha \cdot l \cdot \beta | \alpha \in (L\backslash\{l\})^* \wedge \beta \in b^*\}$$

A.1.14 *Required* message with strict operator and future unwanted message constraint:

$$[[(<r:, a, C, C', \lambda, \bullet b>, \text{strict})]]^{VC} = \{l \cdot \beta | \alpha \in (L\backslash b)^*\}$$
$$[[(<r:, a, C, C', \lambda, \bullet b>, \text{strict})]]^{UDC} = \{\alpha\}$$
$$[[(<r:, a, C, C', \lambda, \bullet b>, \text{strict})]]^{IVT}$$
$$= \{\alpha\} \bigcup \{l \cdot \beta\}, \quad \text{where} \quad (\alpha \in (L\backslash\{l\})^*) \wedge (\beta \in b^*)$$

A.1.15 *Required* message with past unwanted chain constraint: (see equation at the bottom of the page)

A.1.16 *Required* message with past wanted chain constraint:

$$[[(<r:, a, C, C', =>g, \lambda>, \text{nop})]]^{VC} = \{\beta_1 \cdot m_1 \cdot \beta_2 \cdot m_2 \cdot \ldots \cdot \beta_n \cdot m_n \cdot \gamma \cdot l | \beta_i \in (L/\{m_i\})^* \wedge 1 \le i \le n \wedge \gamma \in (L/\{l\})^*\}$$

$[[(<r:, a, C, C', =>g, \lambda>, \text{nop})]]^{UDC} = \{\beta_1\} \bigcup_{\{} \beta_1 \cdot m_1 \cdot \beta_2\} \bigcup_{\ldots} \bigcup_{\{} \beta_1 \cdot m_1 \cdot \beta_2 \cdot \ldots \cdot \beta_{n-1} \cdot m_{n-1} \cdot \gamma\}$, where $(\beta_i \in (L/\{m_i\})^* \wedge (1 \le i \le n)) \wedge (\gamma \in (L/\{\alpha\})^*)$

$[[(<r:, a, C, C', =>g, \lambda>, \text{nop})]]^{IVT} = \{\beta_1\} \bigcup \{\beta_{1'} \cdot m_1 \cdot \beta_2\} \bigcup_{\ldots} \bigcup_{\{} \beta_{1'} \cdot m_1 \cdot \beta_{2'} \cdot \ldots \cdot \beta_{n'} \cdot m_n \cdot \gamma\}$, where $(\beta_i \in (L/\{m_i\})^\infty \wedge \beta_i' \in (L/\{m_i\})^* \wedge (1 \le i \le n)) \wedge (\gamma \in (L/\{l\})^\infty)$.

A.1.17 *Required* message with future unwanted chain

$$[[(<r:, a, C, C', \neq>g, \lambda>, \text{nop})]]^{VC} = \{\beta_1 \cdot l\} \bigcup \{\beta_1 \cdot m_1 \cdot \beta_2 \cdot l\} \bigcup \{\beta_1 \cdot m_1 \cdot \beta_2 \cdot m_2 \cdot \cdots \cdot \beta_{n-1} \cdot m_{n-1} \cdot \gamma \cdot l\}$$

where $(\beta_i \in (L\backslash\{m_i\})^* \wedge 1 \le i \le n-1) \wedge (\gamma \in (L\backslash\{l\})^*)$

$$[[(<r:, a, C, C', \neq>g, \lambda>, \text{nop})]]^{UDC} = \{\beta_1\} \bigcup \{\beta_1 \cdot m_1 \cdot \beta_2\} \bigcup \cdots \bigcup \{\beta_1 \cdot m_1 \cdot \beta_2 \cdot \cdots \cdot \beta_{n-1} \cdot m_{n-1} \cdot \gamma\}$$

where $((\beta_i \in (L\backslash\{m_i\})^* \wedge (1 \le i \le n)) \wedge (\gamma \in (L\backslash\{a\})^*)$

$$[[(<r:, a, C, C', \neq>g, \lambda>, \text{nop})]]^{IVT} = \{\beta_1 \cdot \gamma\} \bigcup \{\beta_1 \cdot m_1 \cdot \beta_2 \cdot \gamma\} \bigcup_{\ldots} \bigcup \{\beta_1 \cdot m_1 \cdot \beta_2 \cdot \cdots \cdot \beta_n \cdot m_n\}$$
$$\bigcup \{\beta_1 \cdot m_1 \cdot \beta_2 \cdot \cdots \cdot \beta_n \cdot m_n \cdot \gamma\}$$

where $((\beta_i \in (L\backslash\{m_i\})^* \wedge (1 \le i \le n)) \wedge (\gamma \in (L\backslash\{l\})^\infty)$

For Research Only

© The Institution of Engineering and Technology 2013

constraint:

$$[[(<r:, a, C, C', \ \neq>g, \lambda>, \text{nop})]]^{\text{VC}}$$
$$= \{\alpha \cdot l\} \bigcup \{\alpha \cdot l \cdot \beta_1 \cdot m_1\} \bigcup \{\alpha \cdot l \cdot \beta_1 \cdot m_1 \cdot \beta_2 \cdot m_2\}$$
$$\bigcup \{\alpha \cdot l \cdot \beta_1 \cdot m_1 \cdot \beta_2 \cdot m_2 \cdot \cdots \cdot \beta_{n-1} \cdot m_{n-1}\}$$
$$\text{where} \quad \alpha \in (L\backslash\{l\})^* \wedge \left(\beta_i \in (L\backslash\{m_i\})^* \wedge (1 \le i \le n)\right)$$

$$[[(<r:, a, C, C', \ \neq>g, \lambda>, \text{nop})]]^{\text{UDC}}$$
$$= \{\alpha\} \bigcup \{\alpha \cdot l \cdot \beta_1\} \bigcup \{\alpha \cdot l \cdot \beta_1 \cdot m_1 \cdot \beta_2\} \bigcup$$
$$\ldots \bigcup \{\alpha \cdot l \cdot \beta_1 \cdot m_1 \cdot \beta_2 \cdot m_2 \cdot \cdots \cdot \beta_{n-1} \cdot m_{n-1}\}$$
$$\text{where} \quad \alpha \in (L\backslash\{l\})^* \wedge (\beta_i \in (L\backslash\{m_i\})^* \wedge 1 \le i \le n)$$

$$[[(<r:, a, C, C', \ \neq>g, \lambda>, \text{nop})]]^{\text{IVT}}$$
$$= \{\alpha\} \bigcup \{\alpha' \cdot l \cdot \beta_1 \cdot m_1 \cdot \beta_2 \cdot m_2 \cdot \cdots \cdot \beta_n \cdot m_n\}$$

A.1.18 *Required* message with future wanted chain constraint:

$$[[(<r:, a, C, C', \lambda, \ =>g>, \text{nop})]]^{\text{VC}}$$
$$= \{\alpha \cdot l \cdot \beta_1 \cdot m_1 \cdot \beta_2 \cdot m_2 \cdot \cdots \cdot \beta_n \cdot m_n | \alpha \in (L\backslash\{l\})^*$$
$$\wedge \beta_i \in (L\backslash\{m_i\})^* \wedge 1 \le i \le n-1$$

$$[[(<r:, a, C, C', \lambda, \ =>g>, \text{nop})]]^{\text{UDC}}$$
$$= \{\alpha\} \bigcup \{\alpha \cdot l \cdot \beta_1\} \bigcup \{\alpha \cdot l \cdot \beta_1 \cdot m_1 \cdot \beta_2\} \bigcup$$
$$\cdots \bigcup \{\alpha \cdot l \cdot \beta_1 \cdot m_1 \cdot \beta_2 \cdot \cdots \cdot \beta_{n-1} \cdot m_{n-1}\}$$
$$\text{where} \quad (\alpha \in (L\backslash\{l\})^*) \wedge (\beta_i \in (L\backslash\{m_i\})^* \wedge 1 \le i \le n-1)$$

$$[[(<r:, a, C, C', \lambda, \ =>g>, \text{nop})]]^{\text{IVT}}$$
$$= \{\alpha\} \bigcup \{\alpha' \cdot l \cdot \beta_1\} \bigcup \{\alpha' \cdot l \cdot \beta_{1'} \cdot m_1 \cdot \beta_2\} \bigcup$$
$$\cdots \bigcup \{\alpha' \cdot l \cdot \beta_{1'} \cdot m_1 \cdot \beta_{2'} \cdot \cdots \cdot \beta_{n-1} \cdot m_{n-1}\}$$
$$\text{where} \quad \alpha \in (L\backslash\{l\})^\infty \wedge \alpha' \in (L\backslash\{l\})^* \wedge \beta_i \in (L\backslash\{m_i\})^\infty$$
$$\wedge \left(\beta_{i'} \in (L\backslash\{m_i\})^* \wedge 1 \le i \le n\right)$$

A.1.19–A.1.23 relate to the denotational semantics of *Fail* messages.

A.1.19 *Fail* message without constraint:

$$[[(<f:, a, C, C', \lambda, \lambda>), \text{nop})]]^{\text{VC}} = \{\alpha | \alpha \in (L\backslash\{l\})^\infty\}$$

$$[[(<f:, a, C, C', \lambda, \lambda>), \text{nop})]]^{\text{UDC}} = \{\alpha | \alpha \in (L\backslash\{l\})^*\}$$

$$[[(<f:, a, C, C', \lambda, \lambda>), \text{nop})]]^{\text{IVT}} = \{\alpha \cdot l | \alpha \in (L\backslash\{l\})^*\}$$

A.1.20 *Fail* message with strict operator:

$$[[(<f:, a, C, C', \lambda, \lambda>), \text{strict})]]^{\text{VC}} = \{\alpha | \alpha \in (L\backslash\{l\})^*\}$$

$$[[(<f:, a, C, C', \lambda, \lambda>), \text{strict})]]^{\text{UDC}} = \{\varepsilon\}$$

$$[[(<f:, a, C, C', \lambda, \lambda>), \text{strict})]]^{\text{IVT}} = \{l\}$$

A.1.21 *Fail* message with past unwanted constraint:

$$[[(<f:, a, C, C', \bullet b, \lambda>, \text{nop})]]^{\text{VC}} = \{\beta\} \bigcup \{\alpha \cdot \gamma\}, \quad \text{where}$$
$$\beta \in b^\infty \wedge \alpha \in (L\backslash b)^* \wedge \gamma \in (L\backslash l)^\infty$$

$$[[(<f:, a, C, C', \bullet b, \lambda>, \text{nop})]]^{\text{UDC}}$$
$$= \left\{\alpha | \alpha \in ((L\backslash b) \bigcap (L\backslash l))^*\right\}$$

$$[[(<f:, a, C, C', \bullet b, \lambda>, \text{nop})]]^{\text{IVT}} = \{\alpha \cdot l | \alpha \in (L\backslash b)^*\}$$

A.1.22 *Fail* message with past unwanted chain constraint: (see equation at the bottom of the page)

$$[[(<f:, a, C, C', \ \neq>g, \lambda>, \text{nop})]]^{\text{VC}}$$
$$= \{\beta_1 \cdot \gamma\} \bigcup \{\beta_1 \cdot m_1 \cdot \beta_2 \cdot \gamma\} \bigcup \cdots \bigcup \{\beta_1 \cdot m_1 \cdot \beta_2 \cdot \cdots \cdot \beta_n \cdot m_n\} \bigcup \{\beta_1 \cdot m_1 \cdot \beta_2 \cdot \cdots \cdot \beta_n \cdot m_n \cdot \gamma\}$$
$$\text{where} \quad (\beta_i \in (L\backslash\{m_i\})^* \wedge (1 \le i \le n-1)) \wedge (\gamma \in (L\backslash\{l\})^\infty)$$

$$[[(<f:, a, C, C', \ \neq>g, \lambda>, \text{nop})]]^{\text{UDC}}$$
$$= \{\beta_1\} \bigcup \{\beta_1 \cdot m_1 \cdot \beta_2\} \bigcup \cdots \bigcup \{\beta_1 \cdot m_1 \cdot \beta_2 \cdot \cdots \cdot \beta_{n-1} \cdot m_{n-1} \cdot \gamma\}$$
$$\text{where} \quad \left(\beta_i \in (L\backslash\{m_i\})^* \wedge (1 \le i \le n-1)\right) \wedge \left(\gamma \in (L\backslash\{l\})^*\right)$$

For Research Only

© The Institution of Engineering and Technology 2013

doi: 10.1049/iet-sen.2012.0047

Table 5 Correspondence between denotational semantics and operational semantics of PSC

denotational semantics	A.1.1	A.1.2	A.1.3	A.1.4	A.1.5	A.1.6	A.1.7	A.1.8	A.1.9
operational semantics	ER1	ER2	ER3	ER4	ER5	ER6	ER7	ER8	ER9
denotational semantics	A.1.10	A.1.11	A.1.12	A.1.13	A.1.14	A.1.15	A.1.16	A.1.17	A.1.18
operational semantics	RR1	RR2	RR3	RR4	RR5	RR6	RR7	RR8	RR9
denotational semantics	A.1.19	A.1.20	A.1.21	A.1.22	A.1.23	A.2.1	A.2.2	A.2.3	A.2.4
operational semantics	FR1	FR2	FR3	FR4	FR5	Def 4	Def 5	Def 6	Def 7
denotational semantics	A.2.5								
operational semantics	Def 8								

A.1.23 *Fail* message with past wanted chain constraint:

$$[[(<f:, a, C, C', =>g, \lambda>, \text{nop})]]^{\text{VC}}$$
$$= \{\beta_1\} \bigcup \{\beta_{1'} \cdot m_1 \cdot \beta_2\} \bigcup \cdots$$
$$\bigcup \{\beta_{1'} \cdot m_1 \cdot \beta_{2'} \cdot \cdots \cdot \beta_{n'} \cdot m_n \cdot \gamma\},$$
where $(\beta_i \in (L\backslash\{m_i\})^{\infty}) \wedge (\beta_{i'} \in (L\backslash\{m_i\})^{*} \wedge (1 \le i \le n)) \wedge (\gamma \in (L\backslash\{l\})^{\infty})$

$$[[(<f:, a, C, C', =>g, \lambda>, \text{nop})]]^{\text{UDC}}$$
$$= \{\beta_1\} \bigcup \{\beta_1 \cdot m_1 \cdot \beta_2\} \bigcup \cdots$$
$$\bigcup \{\beta_1 \cdot m_1 \cdot \beta_2 \cdot \cdots \cdot \beta_{n-1} \cdot m_{n-1} \cdot \gamma\}$$
where $(\beta_i \in (L\backslash\{m_i\})^{*} \wedge (1 \le i \le n)) \wedge (\gamma \in (L\backslash\{l\})^{*})$

$$[[(<f:, a, C, C', =>g, \lambda>, \text{nop})]]^{\text{IVT}}$$
$$= \{\beta_1 \cdot m_1 \cdot \beta_2 \cdot m_2 \cdot \cdots \cdot \beta_n \cdot m_n \cdot \gamma \cdot l | \beta_i \in (L\backslash\{m_i\})^{*}$$
$$\wedge 1 \le i \le n \wedge \gamma \in (L\backslash\{l\})^{*}$$

9.1.2 Compositional semantics: A.2.1 'Merge' operator is used to compose the denotational semantics of two continuous messages. The semantics can be defined as follows: (see equation at the bottom of the page)

$$[[\text{msg}_1 \cdot \text{msg}_2]]^{\text{UDC}} = [[\text{msg}_1]]^{\text{UDC}} \bigcup [[\text{msg}_2]]^{\text{UDC}}$$

$$[[\text{msg}_1 \cdot \text{msg}_2]]^{\text{IVT}} = [[\text{msg}_1]]^{\text{IVT}} \bigcup [[\text{msg}_2]]^{\text{IVT}}$$

A.2.2 'Alternative' operator is used to compose the denotational semantics of two alternative messages. The semantics can be defined as follows:

$$[[\text{msg}_1 \otimes \text{msg}_2]]^{\text{VC}} = [[\text{msg}_1]]^{\text{VC}} \bigcup [[\text{msg}_2]]^{\text{VC}}$$

$$[[\text{msg}_1 \otimes \text{msg}_2]]^{\text{UDC}} = [[\text{msg}_1]]^{\text{UDC}} \bigcup [[\text{msg}_2]]^{\text{UDC}}$$

$$[[\text{msg}_1 \otimes \text{msg}_2]]^{\text{IVT}} = [[\text{msg}_1]]^{\text{IVT}} \bigcup [[\text{msg}_2]]^{\text{IVT}}$$

Based on Merge and Alternative, the operators Alt, Par and Loop can be defined as follows:

A.2.3 *Alt operator*: $\text{Alt}(\omega^{i_1,j_1}, \omega^{i_2,j_2}, \ldots, \omega^{i_r,j_r}, r)$, where $\omega^{i_1,j_1}, \omega^{i_2,j_2}, \ldots, \omega^{i_r,j_r}$ means r part alternatives. The definition of Alt operator can be composed by Merge and Alternative composition.
$[[\text{Alt}(\omega^{i_1,j_1}, \omega^{i_2,j_2}, \ldots, \omega^{i_r,j_r}, r)]]^{\text{trace}} = [[[[\omega^{i_1,j_1}]^{\text{trace}} \otimes [[\omega^{i_2,j_2}]] \otimes \cdots \otimes [[\omega^{i_r,j_r}]].^{\text{trace}}]]^{\text{trace}}$, where

$$[[\omega^{i_k,j_k}]]^{\text{trace}} = \left[[\text{msg}_{i_k}]\right]^{\text{trace}} \otimes \left[[\text{msg}_{i_{k+1}}]\right]^{\text{trace}} \otimes \cdots$$
$$\otimes \left[[\text{msg}_{j_k}]\right]^{\text{trace}} (1 \le k \le r)$$

A.2.4 *Par operator*: $\text{Par}(\omega^{i_1,j_1}, \omega^{i_2,j_2}, \ldots, \omega^{i_r,j_r}, r)$, where $\omega^{i_1,j_1}, \omega^{i_2,j_2}, \ldots, \omega^{i_r,j_r}$ means r parts to parallel. The definition of Par operator can also be composed by Merge and Alternative composition.

$$[[\text{Par}(\omega^{i_1,j_1}, \omega^{i_2,j_2}, \ldots, \omega^{i_r,j_r}, r)]]^{\text{trace}} = [[[[\omega^{i_1,j_1}]]^{\text{trace}}$$
$$\otimes [[\omega^{i_2,j_2}]] \otimes \cdots \otimes [[\omega^{i_n\text{um(Par)},j_n\text{um(Par)}}]]^{\text{trace}}]]^{\text{trace}}, \text{ where}$$

$$[[\omega^{i_k,j_k}]]^{\text{trace}} = \left[[\text{msg}_{i_k}]\right]^{\text{trace}} \otimes \left[[\text{msg}_{i_{k+1}}]\right]^{\text{trace}} \otimes \cdots$$
$$\otimes \left[[\text{msg}_{j_k}]\right]^{\text{trace}} (1 \le k \le \text{num(Par)})$$

A.2.5 *Loop operator*: Loop(ω, m, n), ω is the loop body and m and n are the lower and upper bound of loop, respectively. The definition of Loop operator can also be composed by Merge and Alternative composition. $[[\text{Loop}(\omega^{i,j}, m, n)]]^{\text{trace}} = [[[[\omega^{i,j}]]^{\text{trace}} \otimes 2[[\omega^{i,j}]]^{\text{trace}} \otimes \cdots \otimes (m - n + 1)[[\omega^{i,j}]]^{\text{trace}}]]^{\text{trace}}$, where

$$[[\omega^{i,j}]]^{\text{trace}} = [[\text{msg}_i]]^{\text{trace}} \cdot [[\text{msg}_{i+1}]]^{\text{trace}} \cdot \cdots \cdot$$
$$[[\text{msg}_j]]^{\text{trace}} (1 \le k \le m - n + 1)$$

$$k\left[[\omega^{i,j}]\right]^{\text{trace}} = \frac{\left[[\omega^{i,j}]\right]^{\text{trace}} \cdot \left[[\omega^{i,j}]\right]^{\text{trace}} \cdot \cdots \cdot \left[[\omega^{i,j}]\right]^{\text{trace}}}{k \text{ times}}$$

$$[[(<f:, a, C, C', \neq>g, \lambda>, \text{nop})]]^{\text{IVT}}$$
$$= [\beta_1 \cdot l] \bigcup \{\beta_1 \cdot m_1 \cdot \beta_2 \cdot l\} \bigcup \{\beta_1 \cdot m_1 \cdot \beta_2 \cdot m_2 \cdots \beta_{n-1} \cdot m_{n-1} \cdot \gamma \cdot l\}$$
where $(\beta_i \in (L\backslash\{m_i\})^{*} \wedge (1 \le i \le n)) \wedge (\gamma \in (L\backslash\{l\})^{*})$

$$[[\text{msg}_1 \cdot \text{msg}_2]]^{\text{VC}} = \{\alpha_1 \cdot \alpha_2 | \alpha_1 \in [[\text{msg}_1]]^{\text{VC}} \wedge \alpha_2 \in [[\text{msg}_2]]^{\text{VC}}\}$$

For Research Only

© The Institution of Engineering and Technology 2013

9.2 Consistency between denotational and operational semantics of PSC

Theorem 1: Given a PSC composed by basic rules and operational rules, the denotational semantics and operational semantics are consistent. In detail, it means that:

- The traces accepted by the glue states of FAs of operational semantics equal the validated continuations defined by denotational semantics.
- The traces accepted by the intermediate states of FAs of operational semantics equal the undecided continuations defined by denotational semantics.
- The traces accepted by the accepting states of FAs of operational semantics equal the invalidated traces defined by denotational semantics.

Proof: The proof is composed of the following three steps:

- In the first step, we prove the consistency between operational semantics and denotational semantics of basic *arrowMsg* (with strict operators or other constraints). This can be proved by the corresponding basic operational semantics to basic denotational semantics. Table 5 shows the corresponding relations between operational semantics and denotational semantics of basic messages. We use A.1.1 and ER1 as an example.

$$\text{A.1.1} \Leftrightarrow \text{ER1}$$

According to the FA generated by ER1, the finite traces accepted by the glue state can be described by the following regular expression: $(!l)^* \cdot l = (L/\{l\})^* \cdot l$, which equals $[[a.1]]^{VC} = \{\alpha \cdot l | \alpha \in (L/\{l\})^*\}$. That is to say, the traces accepted by the glue states of FAs of operational semantics equal the validated continuations defined by denotational semantics.

According to the FA generated by ER1, the finite traces accepted by the intermediate state can be described by the following regular expression: $(!l)^\infty = (L/\{l\})^\infty$, which equals $[[a.1]]^{UDC} = \{\alpha | \alpha \in (L/\{l\})^\infty\}$. That is to say, the traces accepted by the intermediate states of FAs of operational semantics equal the undecided continuations defined by denotational semantics.

Other rules can be proved in a similar way.

- The second step is to prove the consistency of Merge and Alternative composition. According to the definition of operational semantics, Merge composition keeps the accepting traces of other states and deletes the glue state of the previous state. Consequently, the accepting traces of the glue states of the newly generated FA is the connection of those in the two FAs. The accepting traces of other states of the new generated FA are the union of those in the two FAs. This is the same as the definition of denotational semantics of Merge. According to the definition of operational semantics, Alternative composition alternatively selects the accepting traces of those in the two states. Consequently, the accepting traces of the states of the new generated FA is the alternative selection of those in the two FAs. This is also the same as the definition of denotational semantics of Alternative.
- The third step is to prove the consistency of operators Alt, Par and Loop. According to the operational semantics, Alt operator first uses Merge composition to generate k alternatives, then repeatedly uses Alternative composition to generate the final FA. Consequently, the validated traces, undecided traces and invalidated traces are alternative selection of these r parts, which is the same definition of denotational semantics. According to the operational semantics, Par operator first uses Merge composition to generate num(Par) alternatives, then repeatedly uses Alternative composition to generate the final FA. Consequently, the validated traces, undecided trace and invalidated trace are alternative selection of these num(Par) parts, which is the same definition of denotational semantics. According to the operational semantics, Loop operator first uses Merge composition to generate one sequence, then repeatedly uses Alternative composition to generate the final FA. Consequently, the validated traces, undecided trace and invalidated trace are alternative selection of these $n-m+1$ parts, which is the same definition of denotational semantics.

After these three steps, we can prove that the PSC specifications composed by basic rules and operational rules have consistent operational and denotational semantics. Consequently, Theorem 1 is proved.

For Research Only

© The Institution of Engineering and Technology 2013

doi: 10.1049/iet-sen.2012.0047