A model-driven approach to performance evaluation of ontology-based service composition

Jia Li, Yunni Xia, Xin Luo

Chongqing Technology and Business Institute, Chongqing, China, wslijia@163.com
School of Computers, Chongqing Univ., China, xiayunni@yahoo.com.cn
School of Computers, Chongqing Univ., China

Abstract

The DAML-based Web Service ontology is one of the major standards for modeling and description of distributed service composition. To analyze the performance of composite service processes specified in DAML-S gives the way to tell whether the process meet the performance requirements, and to choose the process with higher efficiency from those with similar function. In this paper, we propose a stochastic-Petri-net-based approach for performance analysis of DAML-S processes, which employs non-markovian-stochastic-petri-nets (SPN) as the intermediate representation. The main innovation of this research includes a translation from DAML-S to non-markovian-stochastic-petri-nets and a performance (using expected-process-normal-completion-time as the metric) analysis method. We also validate the correctness of the approach in the case study by showing 90% confidence intervals obtained from experimental results cover corresponding theoretical prediction values.

Keywords: Service composition, DAML-S, Petri net

1. Introduction

Recently, various efforts have been devoted to propose expressive description languages capable of modeling composite service behavior. The DAML-S [2] based Web Service ontology is one of the major efforts in this direction. DAML-S is a computer-interpretable semantic mark-up language, where, for the first time, ontology-based descriptions of service functionality and of interaction service behavior coexist.

Many researches have been done in formalization and functional analysis of DAML-S. For example, researches based on Petri net [3,4,5,6], process algebra [7,8,9,10] are introduced to formally capture behavioral patterns of ontology-based service composition and build upon a large body of theoretical results as well as techniques for verifying formal and functional properties. However, research in nonfunctional properties of ontology-based service composition is very limited, especially its performance (to our knowledge, this paper is the first contribution addressing performance evaluation of DAML-S based service composition).

Our aim in this paper is therefore the development of a methodology for performance analysis of composite services specified in DAML-S, with special attention to timed and probabilistic aspects. We introduce a novel model-driven approach which employs stochastic-petri-nets (SPN) as the intermediate representation. The main innovation of this research includes a complete translation from DAML-S to SPN and a performance evaluation. To validate our approach, we also obtain experimental results in the case study and show that confidence intervals derived from experimental records perfectly cover corresponding theoretical values.

2. A brief introduction to DAML-S

DAML-S is defined as a W3C standard to provide a computer-interpretable description of the services, service access and service composition using DAML ontologies. Building upon SOAP and WSDL, the DAML-S services can be dynamic executed on the Web. DAML-S models the upper ontology for services from three perspectives: ServiceProfile, Service Grounding, and ServiceModel. The ServiceProfile provides a high-level description of a service and its provider for advertising, requesting and matchmaking the service, including the service brief description,
service functionalities, and functional attributes. The ServiceGrounding defines the mapping from abstract representation to concrete specification which specifies the details of how to access the service such as protocol and message formats, serialization, transport, and addressing. The ServiceModel describes the service capabilities to enable service invocation, composition, monitoring, etc. Two components are used to define the DAML-S process model: the Process Ontology describes the service properties including inputs, outputs, preconditions and effects; and the Process Control Ontology describes the process state such as initial activation, execution and completion. The ServiceModel of DAML-S is modeled as a workflow of processes including atomic, simple and composite processes. Each composite process holds a Control Construct. The Constructs can contain each other recursively, so the composite process can model all of the possible workflows of WS.

3. Model mapping of DAML-S

In this section, we introduce a set of translation rules to describe the control flow semantics of DAML-S processes and map DAML-S constructs and elements into equivalent stochastic Petri net representations. Since the syntax of DAML-S is too vast, we restrict it to a subset of DAML-S, only considering elements directly related with the control flow evolution (DAML-S elements such as input-binding, output-binding, data manipulation, boolean condition evaluation, preconditions and results are abstracted away and omitted). This translation captures the flow of activity execution of the participants of a service composition and their interactions.

DAML-S process is a specification of the ways a client may interact with a service. An atomic process is a description of a service that expects one (possibly complex) message and returns one (possibly complex) message in response. A composite process is one that combines multiple sub processes. An atomic process is executed through the <perform> construct. Atomic process corresponds to an action a service can perform in a single interaction and which can be executed in a single step by sending and receiving appropriate messages. Also, an atomic process has no sub-process. An atomic process can have any number of inputs (through the <hasInput> element), representing the information that is required. It can have any number of outputs (through the <hasOutput> element), the information that the process provides to the requester. There can be any number of preconditions, which must all hold in order for the process to be successfully invoked. Finally, the process can have any number of effects. Outputs and effects can depend on conditions that hold true of the world state at the time the process is performed. The main operation of atomic process is to contact and invoke partner Web Services through the grounding mechanism (the grounding maps the specification of an atomic process into a concrete executable service components described by WSDL).
According to the discussion above, an atomic process in DAML-S is translated into the SPN model given in Fig.1. In Fig.1, the started and completed places indicate the initial and completed states of the process, respectively. The soap.d timed transition denotes the duration needed to establish a successful SOAP connection with the operation defined in the WSDL specification. timer denotes the timeout duration. Because the SOAP connections are prone to failure (caused by message loss, for instance), the soap_f transition is used to capture the SOAP failure and it directly marks failed (in gray) to indicate the unsuccessful invocation. If no SOAP failure occurs and the SOAP delay exceeds the timeout duration, the invocation of the WSDL operations is prevented and the failed place is marked. Otherwise, the timeout transition is prevented and the invocation of the external operations is permitted and the finished place is marked. In this case, the operations which implements the atomic process can either be faulty (by firing the ivk_f transition and marking the failed place) or successful (by firing the complete transition and marking the completed place).

All composite processes bottom in atomic processes. The <composedOf> property describes the control flow and the data flow of sub-processes with in a composite process, yielding constraints on the ordering and conditional execution of these sub-processes. Composite processes can be organized by <sequence>, <split>, <split-join>, <choice>, <any-order>, <if-then-else>, <repeat-while>, and <repeat-until> patterns. We start with the <sequence> pattern. In this pattern, a list of control constructs is executed sequentially. The translated GSPN representation of this process is given in Fig.2. In Fig.2, we assume there are only two sub processes, namely P1 and P2. It is worth noting that P1 and P2 can have input and output conditions associated if they are atomic processes but corresponding input/output places (in and out in Fig.1) are not shown in Fig.2 because they are not part of the process itself. The failure mode of <sequence> is simply implemented by propagating inner failures of P1,2 to the level of <sequence> itself, through immediate transitions from failed1,2 to failed.
The `<choice>` process prescribes that a single one from a given bag of including sub processes (specified by the `<components>` property) is executed. As shown in Fig.3, the choice construct includes two selective branches of $P_1$ and $P_2$, organized by an XOR selective construct. Selecting and completing either branch would allow the composite process to finish. The failure mode of `<choice>` is implemented in a similar way as that of the `<sequence>` process.

![Fig.9 SPN model of repeat-until process](image)

The `<split>` process prescribes that included branches are executed in parallel. It completes as soon as all of its component processes have been scheduled for execution and does not wait for completion of those branches. The translation of the `<split>` activity are shown in Fig.4.

The `<split+join>` process is also used for concurrent execution of branches but is intrinsically different from the `<split>` process. It consists of concurrent execution of a bunch of process components with a barrier synchronization. That is, it completes when all of its component processes have completed. The translation rule for the `<split+join>` process is given in Fig.5.

The `<if-then-else>` process is a control construct associated with a Boolean decision. If the condition is satisfied, the true branch (the `<then>` branch) is selected and executed, otherwise the false branch (the `<else>` branch). The `<if-then-else>` process is accomplished when its selected branch is completed. The translation rule is given in Fig.6, where immediate transitions true/false denote the true/false evaluation of the Boolean condition.

The `<any-order>` process allows the sub processes to be executed in some unspecified order but not concurrently or in an interleaved way. Execution and completion of all branches are required. As shown in Fig.7, the execution of branches in an any-order process cannot overlap and all branches must be executed. Place single and bidirectional arcs from single guarantee that $P_1$ and $P_2$ are not executed concurrently.

Both `<repeat-while>` and `<repeat-until>` processes support iterative execution of included sub process. They keep iterating until a condition becomes false or true, following the familiar programming language conventions. `<repeat-while>` tests for the loop condition. If the condition is false and does the operation if the condition is true, then loops. `<repeat-until>` does the operation, tests for the condition, exits if it is true, and otherwise loops. Thus, `<repeat-while>` may never execute its sub process, whereas `<repeat-until>` always executes at least once. Fig.7-8 show the SPN translation of the two patterns, where the immediate transition back leads the control flow back to the beginning, while the immediate transition skip leads the control flow out.

4. A case study

In this section, we employ the frequently used CongoProcess sample given in [11] to conduct a case study. The sample is translated to the SPN model given in Fig.10. In this figure, the DAML-S elements not used by translation rules (e.g., the input-binding, the output-binding, the data manipulation, the boolean condition evaluation) are not shown. The FullCongoBuy process is the uppermost composite process of the sample. It is organized by an `<any-order>` process and composed of an atomic process, LocateBook, and a composite process, OrderManagement. The OrderManagement process includes two composite processes, namely CongoBuyBook and UserInfoRetrieval. The UserInfoRetrieval process implements a sequential process and includes two atomic processes, namely LoadUserProfile and ValidateUserEmail. The CongoBuyBook process also implements a sequential processes and includes a composite process, BuySequence. The BuySequence
process implements a sequential process and includes an atomic process, PutInCart, and a composite process, SignInAndSpecify. The SignInAndSpecify process implements a <split-join> process and includes two atomic processes, namely SpecifyPaymentMethod and ShipmentManagement.

Fig.10 SPN model of the FullCongoBuy process

5. Performance evaluation
In our research, we used \( EPNCT \) (expected process normal completion time) as the metric for service performance. From the view point of Petri net representation of the DAML-A process, \( EPNCT \) denotes the duration that the initial marking takes to finally reaches successful absorbing markings, which indicates that the outmost normal completion state is marked and that none of the gray places are marked. For example, in Fig.10, a successful absorbing marking is supposed to be a marking where the outmost completed is marked and none of gray faulty places are marked.

\( EPNCT \) is calculated in a recursive way, traversing through all reachable states (reachable markings). The state space is a homogeneous continuous Markov chain and its infinitesimal generator matrix \( Q \) is calculated in Eq.1.

\[
q_{i,j} = \begin{cases} 
\lambda(td_i) \times \prod_{t_i \in TISET} pe(t_m) & \text{if } s_i \xrightarrow{td_i,TISET} s_j \\
- \sum_{1 \leq r \leq |S|, r \neq i} q_{i,r} & \text{if } i = j \\
0 & \text{else}
\end{cases}
\]

where \( \lambda(td_i) \) denotes the execution rate of the transition \( td_i \), \( |S| \) denotes the number of states in the state space and \( q_{i,j} \) denotes the transition rate from state \( s_i \) to \( s_j \). Relation \( s_i \xrightarrow{td_i,TISET} s_j \) implies that \( s_j \) is the resulting state of \( s_i \) if timed transition \( td_i \) and all immediate transitions in the set of \( TISET \) fire.

Note that, the execution rate of the timed transition can be approximately calculated as

\[
\lambda(t_i) \approx \frac{1}{\text{Mean historical running delays of the activity}}
\]

To evaluate \( EPNCT \), we first have to evaluate the expected duration for each marking to reach the absorbing marking of normal completion, \( EDT(i) \). We have

\[
EDT(i) = \begin{cases} 
0 & \text{if } m_i \text{ is a normal completion marking} \\
\infty & \text{if } m_i \text{ is a faulty completion marking} \\
\infty & \text{if } EDT(j) = \infty \text{ for all } s(j) \in IMS(i) \\
-\frac{n}{q_{i,i}} + \sum_{1 \leq k \leq |S|, k \neq i, EDT(k) < \infty} \frac{q_{i,k} \times EDT(k)}{TP(i)} & \text{else}
\end{cases}
\]

where \( IMS(i) \) denotes the set of immediate succeeding markings of \( s_i \), \( \frac{n}{-q_{i,j}} \) denotes the expected elapsed duration of the marking \( m_i \), and \( TP(i) \) is an intermediate variable given by

\[
TP(i) = \sum_{1 \leq k \leq |S|, k \neq i, EDT(k) < \infty} q_{i,k} = -q_{i,i}
\]

Eq.3 implies that the \( EDT \) of a certain marking is its expected elapsed duration plus the weighted \( EDT \) of its immediately succeeding markings (not including faulty completion markings and those that deterministically lead to erroneous completion markings). For brevity, the proof of Eq.3 is omitted.

Based on observations above, \( EPNCT \) is calculated as \( EDT \) of the initial marking

\[
EPNCT = EDT(0)
\]
To prove the feasibility and accuracy of our model, we execute the DAML-S sample given in section three on the OWL-S API tool [12] and conduct a confidence-interval-analysis. The OWL-S API tool is a Java API for programmatic access to read, execute and write DAML-S/OWL-S service descriptions. The API provides an execution engine that can invoke atomic processes with WSDL groundings, and composite processes that uses DAML-S control constructs. It also provides log files on the starting time and ending time of each process and its termination status.

The six atomic processes are implemented by operations specified by the WSDL grounding document at http://www.daml.org/services/DAML-s/1.2/CongoGrounding.wsdl. The timeout value for all WSDL invocations is 250ms. Using the soapUI test tool, we also obtain the SOAP delay data of the six operations through repetitive SOAP invocations. Using these test records, we generate the histogram charts in Fig. 11-16.

Note that the histogram charts shown above are used to derive executions of the \( \text{soap}_d_{1,6} \) timed transitions in Fig. 10, using Eq. 2.

We also obtain the SOAP failure connection rates of the six atomic processes as 0.0042/0.0019/0.0068/0.0075/0.0054/0.0048. That is to say, firing probabilities of \( \text{pe}(\text{soap}_f_{1,6}) \) are 0.0042/0.0019/0.0068/0.0075/0.0054/0.0048. Moreover, we obtain the service failure rates
of the six invoked operations as 0.0035/0.0014/0.0032/0.0078/0.0053/0.0025. That is to say, firing probabilities of $pe(ivk, f_{1,6})$ are 0.0035/0.0014/0.0032/0.0078/0.0053/0.0025.

![Fig.17](image.png)

Fig.17. The histogram chart of normal-completion-time

Based on the runtime log of DAML-S API engine, we obtain the experimental normal-completion-time data and illustrate it in Fig.17. Employing normal distribution as the fitting function, we use data in Fig.17 to derive the 90% confidence (employing the normal distribution as the fitting function) interval of $EPNCT$ as [1724.1ms, 1743.9ms], while the theoretical estimates of $EPNCT$ based on the SPN approach is 1727.7ms. The coverage of theoretical value by its corresponding confidence interval suggests the correctness and good accuracy of our approach.

7. Conclusions and further study

In this research, we introduce a stochastic approach for integrating performance analysis into service composition described by ontology-based composition language, DAML-S. The proposed approach employs SPN as the intermediate representation and bases itself on translation rules which can map DAML-S elements into equivalent SPN representations. To validate the approach, we also obtain experimental performance results in the case study and derive 90% confidence intervals. The coverage of theoretical prediction values by corresponding confidence intervals suggests that our approach is validated by experiments.

8. References