Building Dynamic Models of Service Compositions with Simulation of Provision Resources*

Dragan Ivanović[1], Martin Treiber[2], Manuel Carro[1], and Schahram Dustdar[2]

[1] Facultad de Informática, Universidad Politécnica de Madrid
idragan@clip.dia.fi.upm.es, mcarro@fi.upm.es
[2] Distributed Systems Group, Technical University of Vienna
{treiber,dustdar}@infosys.tuwien.ac.at

Abstract. Efficient and competitive provision of service compositions depends both on the composition structure, and on planning and management of computational resources necessary for provision. Resource constraints on the service provider side have impact on the provision of composite services and can cause violations of predefined SLA criteria. We propose a methodology for modeling dynamic behavior of provider-side orchestration provision systems, based on the structure of orchestrations that are provided, their interaction, statistically estimated run-time parameters (such as running time) based on log traces, and the model of resources necessary for orchestration provision. We illustrate the application of our proposed methodology on a non-trivial real world example, and validate the approach using a simulation experiment.

Keywords: Service Compositions, Business Process Modeling, Quality of Service, Simulation.

1 Introduction

Service compositions allow organizations to develop complex, cross-organizational business processes by reusing existing services, and are thus attractive for service providers and service consumers alike. Service compositions have been studied thoroughly over recent years and different models to define service compositions have emerged [1]. Approaches like BPEL [2] or YAWL [3] define service compositions in a top down manner using specific notation. At the same time, abstract service composition models include different strands of Petri Nets [4] and process calculi [5].

Key to business usability of service compositions is conformance of their Quality of Service (QoS) attributes with Service-Level Agreements (SLA). Both are intimately related to monitoring and adaptation capabilities [6]. From the computational point of view, resource utilization management, especially the ability to scale computational resources to the expected level of demand, may have drastic impact on response time and failure rates.

* The research leading to these results has received funding from the European Community Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube). Dragan Ivanović and Manuel Carro were also partially supported by Spanish MEC project 2008-05624/TIN *DOVES* and CM project P2009/TIC/1465 (*PROMETIDOS*).

J. Parsons et al. (Eds.): ER 2010, LNCS 6412, pp. 288–301, 2010.

In this paper we take the approach of modeling service compositions with dynamic, continuous-time models that are usually found in applications of system dynamics [7]. We extend the previous work on applying system dynamics to (atomic) service provision management [8], by deriving the quantitative indicators for a service provision chain (number of executing instances, invocation and failure rates) from the structure of a particular composition being provided, as well as from a model of computational resources involved in provision. Usefulness of the system dynamics approach for generating simulators of different (and potentially complex and "non-standard") *what-if* scenarios that reflect interesting situations in provider's environment, has been already studied [9]. Such simulators can be used as a basis for developing and validating provision management policies for service-oriented systems.

We propose an approach that utilizes a common (composition-independent) service provision framework that divides the modeling concern into the composition and the computational resource parts. The former has been studied in several interesting, but specific, cases [9,10], and some examples of a more systematic approaches to building such dynamic composition models based on QoS constraints have been demonstrated [11]. Our intention is to propose a generic method of converting descriptions of orchestrations in the form of a place-transition networks (PT-nets) [12] into dynamic models, in a manner that ensures composability of orchestrations within choreographies. We believe that such automatic dynamic model generation is a prerequisite for practical application.

2 Motivating Example

To illustrate the challenges of our proposed approach, consider the following service composition example. An SME offers a data clearance service, which is the process of filtering arbitrary customer data (e.g., consumer or company addresses). The data cleansing removes duplicates and, if necessary, corrects data using the SME's internal (consumer/company) database. An overview of the service composition is given in Figure 1.

In the first step (*Format Service*), the data set is transformed into a format that is accepted by the services that are used for data cleansing. This is supported by a set of auxiliary tools, like spreadsheets or text editors which allow users to check the content and to apply modifications to the content. Examples include the rearrangement of data columns or the deletion of columns. In the second step, a service (*Data Checker Service*) is used to determine whether a data record represents consumer data or company data. Afterwards, a *Search Service* is used to search the company's database for matching data. If a data record can be found, the customer data record is marked with a unique numeric identifier and the result is stored using a *Storage Service*. If no matching datum is found, the data record is used to create a new entry in the SME database with an unique identifier (*Add Service*). In the case of multiple hits, the data is checked again (*Check Service*) using additional resources like online databases to look for additional data to confirm the identity of a data record and either to create a new entry in the SME database or to assign an unique identifier manually. These activities are controlled

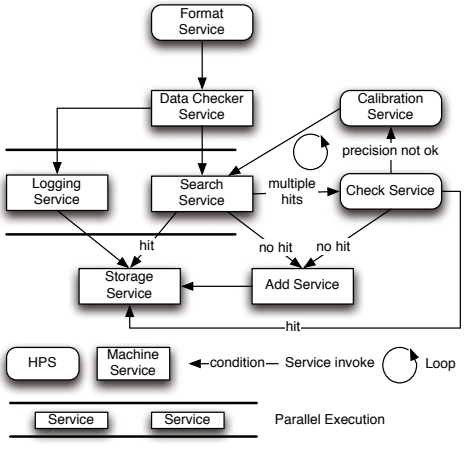


Fig. 1. Overview of data cleansing process

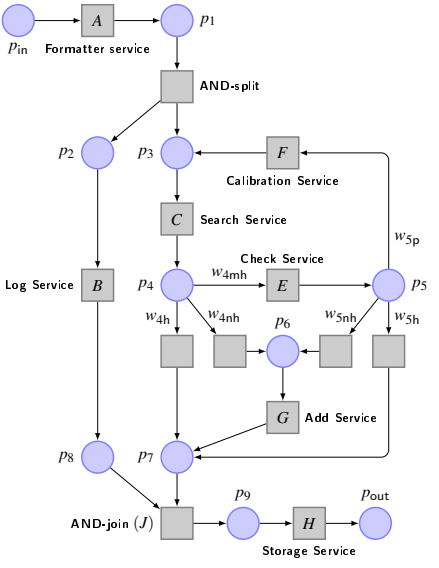


Fig. 2. A PT-net representation of workflow from Fig. 1

by employees who supervise the search service. If the observed quality of the search result is not satisfying (e.g., low precision of the search result), a re-calibration of the search service is done manually and the re-calibrated *Service Service* is invoked again. In parallel, a *Logging Service* is executed to log service invocations (execution times and service precision).

Figure 2 shows a representation of the cleansing process workflow from Figure 1 in the shape of a PT-net. Services from the workflow are marked as transitions ($A \ldots H$), and additional transitions are introduced for the AND-split/join. Places represent the arrival of an input message (p_{in}), end of the process (p_{out}), and activity starting/ending. The places that represent branches to mutually exclusive activities (p_4 and p_5) have outgoing branches annotated with non-negative weight factors that add up to 1; e.g., the branches $w_{4\mathsf{h}}$, $w_{4\mathsf{nh}}$ and $w_{4\mathsf{mh}}$ of p_4, which correspond to the *single hit*, *no hits* and *multiple hits* outcomes of the search service (C), respectively.

As indicated by the example, the actual execution time of the service composition depends on various factors, like the execution time of the human-provided services. Based on these observations, we can summarize the challenges of the working example as follows:

– Unpredictable content. The content of the customer data varies in terms of data structure, data size, and quality. For instance, there might be missing and/or wrong parts of addresses or wrong names and misplaced column content which can cause extra efforts.

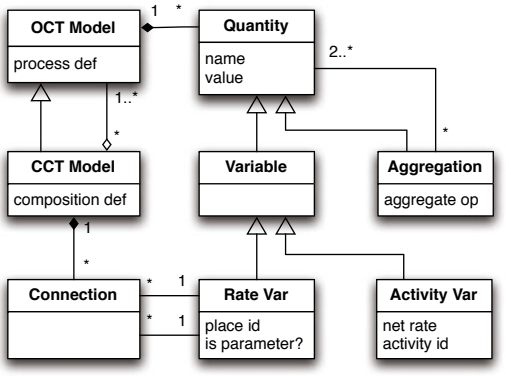


Fig. 3. Conceptual view of composition CT model

- Unpredictable manual intervention. Depending on the data size and the data quality the precision of the search result differs. This requires manual calibration of search profiles during the execution of the task.
- Customer specific details regarding the provided and expected data format need also to be considered. For instance a customer provides excel spreadsheets, while another costumer favors ASCII files with fixed length columns and another provides a XML structure.
- Delegation. Employees might delegate parts of their activities to other employees due to time constraints.

All these challenges arise because of resource constraints (e.g., number of employees that work on a given task) and need to be regarded when service compositions are modeled. The provider may be interested not only in maximal levels of required resources, but may also ask, given an input request scenario, *when* the peak loads can be expected to occur, how long it takes for resource loads to stabilize after a change in input regime, and what resource up/down scaling policies are reasonable.

3 Conceptual Dynamic Modeling Framework

3.1 Conceptual Composition Model

We start with a conceptual model for dynamic representation of service compositions (orchestrations and choreographies). The goal of these models is to represent how the numbers of executing activites and rates of activity-to-activity transitions vary over time. Figure 3 presents a conceptual famework for the continuous-time (CT) composition modeling. The fundamental building block of an *orchestration* CT (OCT) *model* is a *variable* that has a time-varying value. Compared to a PT-net model, *activity variables* are attached to transitions inside, and their value is the expected number of executing instances of a given activity at each point in time. In our example, each service in the data cleansing process would be represented with an activity variable, that shows the expected number of concurrently executing instances of the service at any moment.

Rate variables correspond to *places*, i.e., the events, in the workflow: arrival of input messages, dispatching of replies, or branching from one activity to another. They represent the number of the corresponding events per unit of time. Rates that come from outside the OCT are called *parameters*. The most common example is the input message rate (number of incoming messages per unit of time), and the reply rate from the model of an invoked service. In our example, besides the input parameter, the rate variables would include any service-to-service transitions, branches, parallel split-join points, and the resulting finishing rate.

Other than variables, *quantities* in an OCT model include *aggregations*, which use some aggregation operator (e.g. sum, minimum, maximum) to combine values of two or more quantities. Examples include: the sum of activity variables for a given set of activity types, the number of invocations of a specific partner service, and the sum of the reply and failure rates. We could, for instance, create an aggregate quantity that sums all activity variables for services in the data cleansing process that are hosted by the same provider. Such aggregate quantity could be used as an indication of the total load on the provider's infrastructure.

The OCT conceptual model can also be applied as a wrapper for services with unknown structure, or those that are not available for analysis. Such "wrapper OCT models" consist of a single activity variable, which stands for the entire service execution, with the input rate (the parameter), the reply rate, and the failure rate.

A *composition* CT (CCT) *model* is an extension of the concept of OCT model, and puts together one or more sub-models, which are connected by pairing their respective input and reply rates. For instance, when an orchestration A invokes an orchestration B, at least two connections are required. The first one connects A's *send* rate to B's input rate, and the second connects B's *reply* rate to the corresponding A's *receive* rate. Additionally, the B's failure rate may be connected to an appropriate slot in A.

A CCT model includes all quantities from the underlying sub-models, but only the unconnected parameters from the sub-models are qualified as parameters of the CCT model. In the rest of the paper, we implicitly assume that the sub-models used in simulations are connected in such a way that the resulting model has a single parameter, its input message rate.

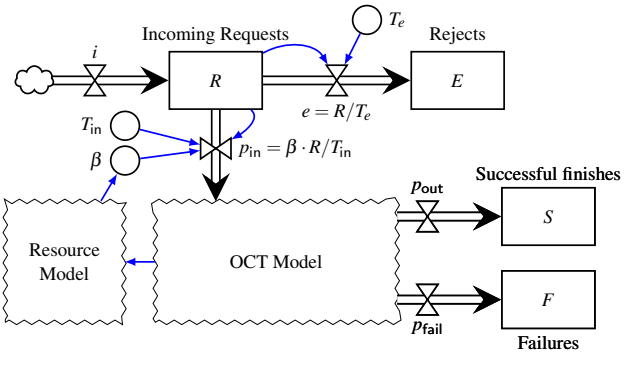


Fig. 4. Simulation setting for the CCT model

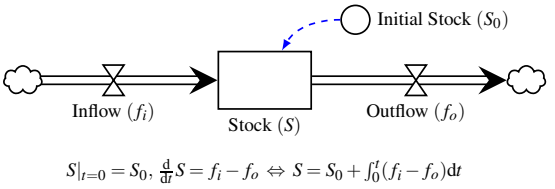


Fig. 5. Stocks, flows and their meaning

3.2 Dynamic Framework Provision Model

Figure 4 shows a view of the framework dynamic provision model that embeds a OCT model and a resource model, using the stock/flow notation [7]. As described in Figure 5, stock/flow diagrams provide a visual notation that corresponds to a part of the system of ordinary differential equations. Rates are represented as flows, while stocks correspond to integrals over the difference between inflows and outflows. Auxiliary variables (circles) represent parameters or intermediate calculations in the model, and arrows depict influences.

The framework dynamic provision model is driven by the input rate (or input regime) i, that models different situations of interest in the environment. Note that this dynamic model (as well as other system dynamic models) is usually not solved analytically, but simulated using numeric integration techniques. This allows experimentation with arbitrary input regime scenarios (i.e., without limiting to the "standard" statistical distributions at the entry). The input regime i fills the stock of received requests R. Some of those requests that are not timely served are rejected (at rate e filling the stock E of rejects). Others, after a period of preparation (modeled by the time constant T_{in}) are fed by the input rate p_{in} into the OCT model. There they execute, and end up either as a successfully finished instance (rate p_{out} filling stock S) or as a failed execution (rate p_{fail} filling stock F).

The relation between the OCT component and the resource model component of the framework model is twofold. First, quantities from the OCT model are used by the resource model to determine the resource load at any moment in time. For instance, sum of all activity variables corresponding to the services in the workflow hosted by the same provider may represent the sum of all threads that need to be available at any moment of time for the orchestration to run. The resource model is, of course, a model of particular resource management policies. Some models can be as simple as to assume no resource limits, other can have "firm" limits, and yet others may include mechanisms for scaling the resources up or down, based on observed trends in the input regime. We will present such a scalable model in Section 4.5; until then, we will tacitly assume the infinite resource case.

The second link is the blocking factor β that regulates the composition input rate p_{in}. β stands for the probability that the system is able to start executing a prepared process instance. In an infinite resource case, $\beta = 1$, but in a more realistic case, as the system load increases beyond the optimal carrying capacity of the provider's infrastructure, the system becomes less able to accept new requests, and β may decrease down to a point of complete denial of service ($\beta = 0$).

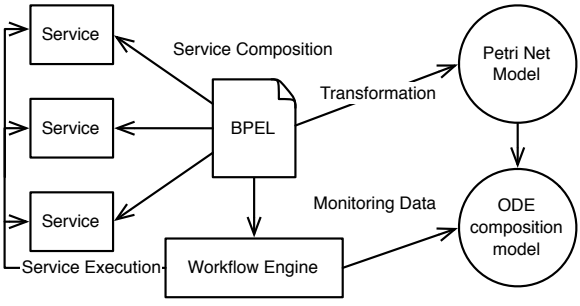


Fig. 6. Overview of the approach

The framework model can be used to produce some basic metrics for the provision system, relative to the input regime i. Assuming that R^+ is the "inflow" part of R, i.e. the sole result of accumulating requests from i (without the effect of e and p_{in}), then if $R^+ > 0$, we may obtain:

- Percentage of all failures and rejects: $(E+F)/R^+$;
- Percentage of all finished orchestrations: $(S+F)/R^+$;
- Percentage of all successful finishes: S/R^+.

4 Automatic Derivation of OCT Models

The derivation of an OCT model for an orchestration follows a series of steps as shown in Figure 6. The overall process starts with the generation of a PT-net model from e.g., an executable BPEL process specification. During the process execution, activity timing and branching probability information is collected, and used to compound (calibrate) the Petri Net model. In the final step, the calibrated Petri Net model is translated into a model based on ordinary differential equations (ODE), whose elements are OCT rates and activity variables.

4.1 Elements of the Petri Net Model

Petri Nets are a formalism frequently used to represent workflows and reason about them [12]. Many standard workflow patterns can be naturally expressed using Petri Nets [13], and there exist numerous tools that allow automatic translation and analysis of service composition languages, such as WS-BPEL [14], and YAWL [3], into a Petri Net representation. Additionally, Petri Net process models can be (partially) discovered from (incomplete) execution log traces. Among many different variations of Petri Nets, we start with simple PT-nets (place-transition networks).

A PT-net is a tuple $\langle P,M,R\rangle$, where P and M are finite non-empty sets of places and transitions, respectively, and $R \subseteq (P \times M) \cup (M \times P)$ is a relation that represents edges from places to transitions and from transitions to places. For any place $p \in P$, we denote the set of input transitions for p as $\bullet p = \{m \in M | (m,p) \in R\}$, and the set of output transitions as $p\bullet = \{m \in M | (p,m) \in R\}$. The sets $\bullet m$ and $m\bullet$ of input and

output places, respectively, for any transition $m \in M$, are defined analogously. For the derivation of an ODE model from the given PT-net, we require that $|\bullet m| > 0$ for each $m \in M$. A marking $s : P \to \mathbb{N}$ assigns to each place $p \in P$ a non-negative integer $s(p)$, known as the *number of tokens*. We say that p is *marked* (under s) if $s(p) > 0$.

A marked place $p \in P$ enables exactly one of its output transitions among $p\bullet$. For a transition m to fire, all places in $\bullet m$ must be marked. When firing, m consumes a token from each $p \in \bullet m$, and when m finishes, it sends a token into each $p \in m\bullet$. In a typical composition setting, tokens are used to describe orchestration *instances*, transitions are used to model *activities*, which may take some time to complete, and places typically represent entry/exit points or pre/post conditions for execution of activities.

4.2 Elements of the ODE Orchestration Model

In a discrete time model of an orchestration provision system, at any moment of time, each running instance of an orchestration has its own marking, and the superposition of these markings gives the aggregate view of the provision system. The aggregate number of tokens $p(t_i)$ in a place $p \in P$ between time steps t_i and t_{i+1} remains stable until $t_{i+1} = t_i + \Delta t_i$, where Δt_i is a discrete time increment, implying instantaneous transitions. In real execution environments, however, activities (transitions) use some definite (sometimes long) amount of time to execute, while tokens stay in places for a very short period of time which is needed by the execution engine to start the next activity.

To build an ODE model based on a PT-net, we consider an idealized execution environment, where the time step Δt_i becomes infinitely small and turns into the time differential $\mathrm{d}t$. Consequently, we can no longer assume that tokens stay in places for definite periods of time, but rather presume they are immediately passed to the destination activities. Therefore, in the CT case, we associate a place $p \in P$ with the *rate* $p(t)$ of tokens passing through p, measured in instances per unit of time.

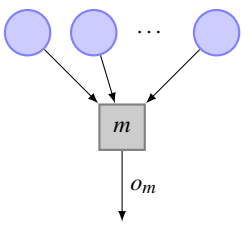
On the other hand, activities are fed by tokens emitted from places at their corresponding rates. For activity $m \in M$, we denote the aggregate number of its currently executing instances at time t with $m(t)$.

In the CT setting, we operate on probabilistic expectations of both rates and activities (transitions). When p has more than one outgoing transition, we use a non-negative real number w_{pm} to denote the expected fraction of $p(t)$ that is passed to $m \in p\bullet$, such that $\sum_{m \in p\bullet} w_{pm} = 1$.

Also, we use exponential decay to model the expected number of executing activity instances. With $m(t)$ we associate a non-negative average execution time T_m. When $T_m = 0$, transition is immediate, and $m(t)$ always remains empty. When $T_m > 0$, we take the usual convenience assumption in dynamic modeling that the running time of individual instances of m obeys Poisson distribution with the average T_m.

The weight factors w_{pm} and the average execution times T_m are assumed to be obtained from execution logs, i.e. the statistical information from previous executions of the orchestrations.

Figure 7 shows a general ODE scheme for a transition $m \in M$ with one or more input places. With single input place, the transition continuously accumulates tokens from the



$m_p(t)$: a new stock $\forall p \in \bullet m$

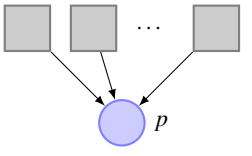
$q_t = \text{argmin}_{p \in \bullet m}\{m_p(t)\}$

$m(t) = m_{q_t}(t)$

$\frac{\mathrm{d}}{\mathrm{d}t} m_p(t) = p(t)w_{pm} - o_m(t)\,, \forall p \in \bullet m$

$$o_m(t) = \begin{cases} m(t)/T_m & T_m > 0 \\ q_t(t)w_{q_t m} & T_m = 0 \end{cases}$$

Fig. 7. ODE scheme for a transition



$p(t) = \sum_{m \in \bullet p}\{o_m(t)\}$

Fig. 8. ODE scheme for a place

input place, and discharges them either instantaneously ($T_m = 0$) or gradually ($T_m > 0$) through $o_m(t)$.

When a transition has more than one input place, its execution is driven by the smallest number of accumulated tokens. At time t, $q_t \in \bullet m$ denotes the the place from which the smallest number of tokens has been accumulated. Because a transition needs to collect a token from all of its input places to fire, the smallest token accumulation $m_{q_t}(t)$ dictates $m(t)$.

When the average execution time $T_m > 0$, the outflow $o_m(t) = m(t)/T_m$ corresponds to exponential decay. When $T_m = 0$, the transition is instantaneous, i.e. $m(t) = 0$, which means that outflow has to balance inflow $q_t(t)w_{q_t m}$ from q_t, therefore keeping $m_{q_t}(t)$ at zero.

Figure 8 shows a general ODE scheme for a place $p \in P$, which is simply a sum of outflows from incoming transitions, assuming that $\bullet p$ is non-empty. Places with an empty set of incoming transitions must be treated as exogenous factors.

4.3 An Example ODE Model

To illustrate the approach to construction of the ODE model, we look at the PT-net representation of our working example, shown in Figure 2. The PT-net model has the starting place p_{in} and the final place p_{out}. Transitions that correspond to invocations of partner services are marked with letters $A..H$, and we assume that the corresponding average execution times $T_A..T_H$ are non-zero. Other transitions are assumed to be instantaneous, and with the exception of the AND-join transition (marked J), they simply propagate their inflow. Places p_4 and p_5 are decision nodes, and their outgoing links are annotated with weight factors corresponding to branch probabilities. With reference to Figure 1, index "h" stands for "hit", "nh" for "no hit", "mh" for "multiple hits", and "p" for "precision not ok." Weights for other (single) place-transition links are

$$\frac{\mathrm{d}}{\mathrm{d}t}A(t) = p_{\mathsf{in}}(t) - p_1(t) \qquad o_F(t) = F(t)/T_F$$
$$p_1(t) = A(t)/T_A \qquad p_6(t) = p_4(t)w_{4\mathsf{nh}} + p_5(t)w_{5\mathsf{nh}}$$
$$p_2(t) = p_1(t) \qquad \frac{\mathrm{d}}{\mathrm{d}t}G(t) = p_6(t) - o_G(t)$$
$$\frac{\mathrm{d}}{\mathrm{d}t}B(t) = p_2(t) - p_8(t) \qquad o_G(t) = G(t)/T_G$$
$$p_8(t) = B(t)/T_B \qquad p_7(t) = p_4(t)w_{4\mathsf{h}} + o_G(t) + p_5(t)w_{5\mathsf{h}}$$
$$p_3(t) = p_1(t) + o_F(t) \qquad \frac{\mathrm{d}}{\mathrm{d}t}J_{p_8}(t) = p_8(t) - p_9(t)$$
$$\frac{\mathrm{d}}{\mathrm{d}t}C(t) = p_3(t) - p_4(t) \qquad \frac{\mathrm{d}}{\mathrm{d}t}J_{p_7}(t) = p_7(t) - p_9(t)$$
$$p_4(t) = C(t)/T_C$$
$$\frac{\mathrm{d}}{\mathrm{d}t}E(t) = p_4(t)w_{4\mathsf{mh}} - p_5(t) \qquad p_9(t) = \begin{cases} p_8(t) & J_{p_8}(t) \le J_{p_7}(t) \\ p_7(t) & J_{p_7}(t) < J_{p_8}(t) \end{cases}$$
$$p_5(t) = E(t)/T_E \qquad \frac{\mathrm{d}}{\mathrm{d}t}H(t) = p_9(t) - p_{\mathsf{out}}(t)$$
$$\frac{\mathrm{d}}{\mathrm{d}t}F(t) = p_5(t)w_{5\mathsf{p}} - o_F(t) \qquad p_{\mathsf{out}}(t) = H(t)/T_H$$

Fig. 9. ODE model for PT-net from Fig. 2

implicitly set to 1. For simplicity, the PT-model does not represent auxiliary computations that in reality take some definite, if small, time to execute.

Figure 9 shows the corresponding ODE model. Some obvious simplifications were applied. For instance, when for a $p \in P$, $\bullet p = \{m\}$ it is not necessary to represent $o_m(t)$ and $p(t)$ separately, so we use the latter. Also, for a place $m \in M$ where $\bullet m = \{p\}$ and $T_m = 0$, we omit the equation for $\frac{\mathrm{d}}{\mathrm{d}t}m(t)$ (which is always 0), and directly propagate $p(t)w_{pm}$ as $o_m(t)$.

The AND-join transition J has two input places, and thus two auxiliary token stocks $J_{p_7}(t)$ and $J_{p_8}(t)$. Since the join is instantaneous, at least one of these two stocks is always zero, and the outflow $p_9(t)$ copies the inflow of the smaller stock.

We assume that the initial marking of the PT-model contains only p_{in}. Consequently, we implicitly assume that the initial condition for all transitions ($A(0)$, $B(0)$, etc.) is zero. Since that place has no input transitions in the model, we assume that $p_{\mathsf{in}}(t)$ is exogenous. Conversely, p_{out} has no output transitions, and we assume that it is the terminal place of the model. The function $p_{\mathsf{out}}(t)$ thus gives the finishing rate of the orchestrations in the ODE model, relative to the start rate $p_{\mathsf{in}}(t)$.

4.4 Asynchronous Composition and Failures

The example PT-net in Figure 2 is simplified as it does not involve asynchronous messaging with partner services, nor accounts for potential failures during service invocations. Both can be built into the model automatically, when translating from a concrete orchestration language with known formal semantics. Here we discuss a way to deal with asynchronicity and faults in a general case.

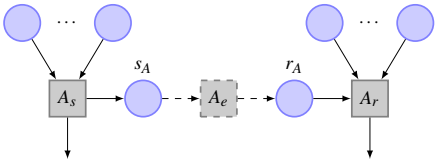


Fig. 10. Asynchronous messaging scheme

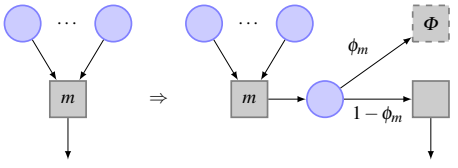


Fig. 11. Failure accounting scheme

Figure 10 shows a usual pattern of asynchronous communication with a partner service A. Transition A_s sends a message to A via a dedicated place s_A, and transition A_r receives the reply through a synchronizing place r_A. The same representation applies to synchronous messaging as well, with A_r directly following A_s, and r_A as its single input place.

In Figure 2, we have combined A_s, s_A, A_e, r_A, and A_r into a single transition A characterized with an overall average execution time T_A. The rate $s_A(t)$ is the *send* rate, to be connected with the input rate parameter of the sub-model of A_e, while $r_A(t)$ is the *reply* rate from the sub-model, to be connected with the *receive* rate in the main OCT model. Examples of "wrapper" sub-models for A_e are: $\{r_A(t) = s_A(t)\}$ (short circuiting, zero time), and $\{r_A(t) = A_e(t)/T_{A_e};\ \frac{\mathrm{d}}{\mathrm{d}t}A_e(t) = s_A(t) - r_A(t)\}$ (black box, definite time).

Failures can be accounted for by introducing failure probabilities ϕ_m for each transition in the PT-net model, and decorating the transitions as shown in Figure 11. Fault handling is represented by Φ. In the simplest case of unrecoverable faults, Φ is an instantaneous transition to a terminal fault place p_{fail}.

4.5 A Sample Resource Model

For a sample resource model, we model threads that execute orchestration activities on the provider's infrastructure. In the sample, shown on Figure 12, we assume that services A, B, G and H from Figure 2 (corresponding to the Formatting, Logging, Adding and Storage services) are "back-end" services hosted by the orchestration provider, so that their each execution occupies a (logical) thread. The number of occupied threads in the resource model is shown as X. The current capacity (available number of threads) is shown as $\hat{X}$, and γ is the degree of utilization. The blocking factor β is 1 if some capacity is free, 0 otherwise.

On the management side, we form a perception X_p of the number of threads required to meet the needs. That perception changes at a rate r_p that is driven by the adjustment time T_p. That is a well known method of approximating formation of perception/trend reporting based on exponential smoothing [7]. Finally, we assume that we

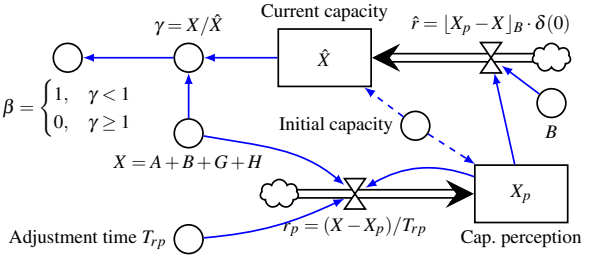


Fig. 12. Thread reasource model

Table 1. Statistical parameters of service substitutes

Service	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
Formatter	15.00	55.50	60.00	63.00	73.00	107.00
Searcher	4.00	4.00	5.00	4.89	6.00	6.00
Checker	0.00	0.00	1.00	0.53	1.00	1.00
Calibrator	59.00	74.00	78.00	81.73	85.50	133.00
Logger	1.00	1.00	1.00	1.00	1.00	1.00
Adder	0.00	0.00	0.50	0.50	1.00	1.00
Storage	1.00	1.00	1.00	1.40	2.00	2.00

can increase/decrease capacity $\hat{X}$ in batches of size B, e.g. corresponding to addition or removal of machines in the provider's cluster. The adjustment rate $\hat{r}$ adds or removes resources when the difference between the target X_p and the current $\hat{X}$ is a multiple of B. The form $\lfloor x \rfloor_B$ is a shortcut for $B \cdot \lfloor x/B \rfloor$; $\delta(0)$ is the Dirac impulse that ensures finite change in infinitesimal time.

5 Simulation Experiment

To validate the proposed approach for automatic construction of OCT models, we have run an experiment based on the example from Section 2. The orchestration was implemented using a service test-bed based on the full service stack, but replacing actual implementations of the partner services (A-H) from Figure 2 with minimal substitutes that introduced time delays in order to mimic execution of their real counterparts. Implementations of the substitute services used the (uniform) random number generation to indicate the result type at branching points (corresponding to places p_4 and p_5 in Figure 2). Along with comparatively greater average execution times, substitutes for human-provided services (Formatting, Calibration, Checking) introduced additional outliers with respect to the basic distribution. Table 1 shows the usual statistical summary of the distributions for simulated execution times of the partner services (all times measured in seconds).

For validation purposes, we simulated the predicted and the actual execution times in the controlled simulation environment. The histogram on Figure 13 shows the distribution of orchestration running times from 30 simulated runs. The median running time was 154, and the average was 214 (seconds).

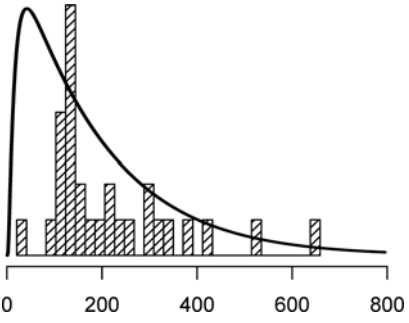


Fig. 13. Distribution of execution times

In order to compare the OCT model prediction of the orchestration running time, we have taken the ODE model from figure 9 and set the input rate $p_{\mathsf{in}} = \delta(0)$, to simulate execution of a single orchestration instance that starts at $t = 0$. Because the orchestration model is lossless (there is no failure rate), the only outgoing rate is p_{out}, which therefore gives the probability distribution over execution times.

The smooth bold curve in Figure 13 shows p_{out} in comparison to the histogram that shows experimental execution times. The median value of the predicted execution time was 142.82 and the average 197.55 (seconds). Although slightly more optimistic than the experimental runs, on the overall the prediction qualitatively fits well with the measured data, especially considering that the prediction does not take into account message passing latencies, and that the experimental setting has used a mix of different statistical distributions for execution times of individual services.

6 Conclusions and Future Work

The approach proposed in this paper can be used for developing dynamic models of service composition provision, based on an automatically derived continuous-time ordinary differential equation model of the target orchestration. The orchestration model is calibrated using empirical estimates of average activity execution times and branching probabilities, obtained from log analysis, event signaling, or other monitoring tools at the infrastructure level. Several dynamic models of orchestrations provided together can be composed in a modular way.

The resulting dynamic model of composition provision can be used for exploring how the provision system reacts to different input rates (requests per unit of time), testing and choosing different resource management strategies and their parameters, in the style of *management flight simulators*. The model output can be used for both quantitative prediction and qualitative assessment of reference modes (growth, oscillation, stagnation, etc.).

Our future work will concentrate on developing tools that allow simultaneous model calibration/simulation using live monitoring data, along with using these data for orchestration process discovery, when the design and its representation in Petri Net form is not given.

References

1. van der Aalst, W.: Don't Go With the Flow: Web Services Composition Standards Exposed. IEEE Intelligent Systems (January/February 2003)
2. Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S.: Business Process Execution Language for Web Services (2003)
3. van der Aalst, W.M.P., ter Hofstede, A.H.M.: Yawl: yet another workflow language. Inf. Syst. 30(4), 245–275 (2005)
4. Esparza, J., Lakos, C. (eds.): ICATPN 2002. LNCS, vol. 2360. Springer, Heidelberg (2002)
5. van der Aalst, W.: Pi calculus versus petri nets: Let us eat "humble pie" rather than further inflate the "pi hype" (2003)
6. Michlmayr, A., Rosenberg, F., Leitner, P., Dustdar, S.: Comprehensive qos monitoring of web services and event-based sla violation detection. In: MWSOC 2009: Proceedings of the 4th International Workshop on Middleware for Service Oriented Computing, pp. 1–6. ACM, New York (2009)
7. Sterman, J.D.: Business Dynamics: Systems Thinking and Modeling for a Complex World. Irwin McGraw-Hill (2000)
8. An, L., Jeng, J.-J.: Web service management using system dynamics. In: ICWS, pp. 347–354. IEEE Computer Society, Los Alamitos (2005)
9. Lee, J.H., Han, Y.S., Kim, C.H.: It service management case based simulation analysis and design: Systems dynamics approach. In: International Conference on Convergence Information Technology, pp. 1559–1566 (2007)
10. Orta, E., Ruiz, M., Toro, M.: A system dynamics approach to web service capacity management. In: European Conference on Web Services, pp. 109–117 (2009)
11. Zhao, H., Tong, H.: A dynamic service composition model based on constraints. In: GCC, pp. 659–662 (2007)
12. Hamadi, R., Benatallah, B.: A petri net-based model for web service composition. In: ADC 2003: Proceedings of the 14th Australasian database conference, Darlinghurst, Australia, pp. 191–200. Australian Computer Society, Inc., Australia (2003)
13. van der Aalst, W., ter Hofstede, A., Kiepuszewski, B., Barros, A.: Workflow patterns. Distributed and Parallel Databases 14(3), 5–51 (2003)
14. WS-BPEL: Business Process Execution Language for Web Services Version 2.0 (April 2007)