11 PEPA Case Study: Web Service Composition

As an example of a realistic case study, we consider a business application which is composed from a number of offered web services. Furthermore there is an access control issue, as it must be ensured that the web service consumer has the necessary authority to execute the web services it requests. A schematic representation of the system is depicted in Fig. 22.

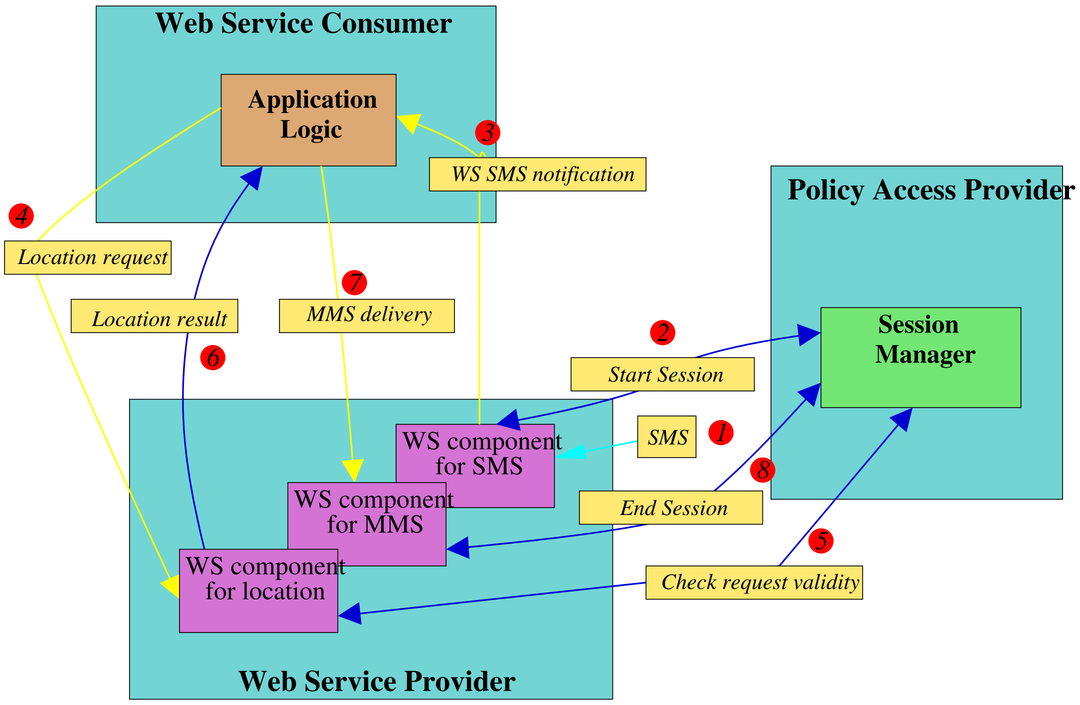

Figure 22: Schematic representation of the web service composition

The scenario is as follows. Several web services are combined to define the business logic of an application. For example, consider an application to find the nearest restaurant for a user and show it on a map. This could involve web services for SMS and MMS handling in addition to the User Location web service. Moreover, a user should not be able to gain access to location information of an arbitary user. This is where the access control aspect becomes important. Therefore, in addition to the requested web services, the web service provider may need to interact with some authorisation component to check that the current user has the correct authority to access the requested information. In adddition the service provider may stipulate some further conditions, such as that only one location request may be made per session:

1. The user activates a service by sending an SMS to a service centre number. This is handled by an appropriate web service.
2. This initiates a *start-session* message to be sent to the Policy Access Provider.
3. A notification is sent to the application that an SMS has arrived.

4. The application requests the user's location from a location web service.
5. The web service contacts the session manager within the policy access provider to check the validity of the request.
6. If the validity check is OK the location web service will return the location to the application which uses it to construct the appropriate map for the user.
7. This is then passed as an MMS to the MMS web service which delivers it to the user.
8. The MMS web service terminates the session with the Session Manager.

We model such a system with the following PEPA model. It has three types of model component, corresponding to the three large rectangles in Figure 22. Note that although the Web Service Provider consists of three distinct elements, we are interested in the session associated with each Web Service Consumer. Each session is associated with an instance of the Web Service Provider. Thus, concurrency is introduced into the model by allowing multiple sessions rather than by representing the constituent web services separately.

11.1 Component *Customer*

The customer's behaviour is simply modelled with two local states. In the first state the customer sends a request to the system via the *getSMS* action. She then waits for a response which triggers the *getMap* transition if it is successful. Thus we associate the user-perceived system performance with the throughput of this action, which can be calculated directly from the steady-state probability distribution of the underlying Markov chain.

$$\begin{aligned} \mathit{Customer} &\stackrel{\mathit{def}}{=} (\mathit{getSMS}, r_1).\mathit{Customer}_1 \\ \mathit{Customer}_1 &\stackrel{\mathit{def}}{=} (\mathit{getMap}, \top).\mathit{Customer} + (\mathit{get404}, \top).\mathit{Customer} \end{aligned}$$

In this model sending either an error message *get404* or the requested map occur at the same rate *r8* and MMS passing between web services is ten times as fast as the communication with the user.

11.2 Component *WSConsumer*

The web service consumer, *WSConsumer*, follows a simple pattern of behaviour. Once it is notified that a session has been started by the user (via SMS message), it initiates a request for the user's current location and waits for a response. If the request was valid, the location is returned and used to compute the appropriate map for the user, which is then sent via an MMS message, using the web service for this.

$$\begin{aligned}
\mathit{WSConsumer} &\stackrel{\mathit{def}}{=} (\mathit{notify}, \top).\mathit{WSConsumer}_2 \\
\mathit{WSConsumer}_2 &\stackrel{\mathit{def}}{=} (\mathit{locReq}, r_4).\mathit{WSConsumer}_3 \\
\mathit{WSConsumer}_3 &\stackrel{\mathit{def}}{=} (\mathit{locRes}, \top).\mathit{WSConsumer}_4 + (\mathit{locErr}, \top).\mathit{WSConsumer} \\
\mathit{WSConsumer}_4 &\stackrel{\mathit{def}}{=} (\mathit{compute}, r_7).\mathit{WSConsumer}_5 \\
\mathit{WSConsumer}_5 &\stackrel{\mathit{def}}{=} (\mathit{sendMMS}, r_9).\mathit{WSConsumer}
\end{aligned}$$

11.3 Component *WSProvider*

As explained above, although the Web Service Provider can be viewed as consisting of three independent web services, the use of sessions restricts a user's access to these services to be sequential. We assume that there is a distinct instance of the component *WSProvider* for each distinct session. As each would be in a distinct thread it is reasonable for there to be concurrency at this level. The activities of the component are as outlined in the scenario above. Note that the *checkValid* action is represented twice, to capture the two possible distinct outcomes of the action. If the check is successful the location must be returned to the Web Service Consumer in the form of a map (*getMap*). However, if the check revealed an invalid request (*locErr*) then an error must be returned to the Web Service Consumer (*get404*) and the session terminated (*stopSession*).

$$\begin{aligned}
\mathit{WSProvider} &\stackrel{\mathit{def}}{=} (\mathit{getSMS}, \top).\mathit{WSProvider}_2 \\
\mathit{WSProvider}_2 &\stackrel{\mathit{def}}{=} (\mathit{startSession}, r_2).\mathit{WSProvider}_3 \\
\mathit{WSProvider}_3 &\stackrel{\mathit{def}}{=} (\mathit{notify}, r_3).\mathit{WSProvider}_4 \\
\mathit{WSProvider}_4 &\stackrel{\mathit{def}}{=} (\mathit{locReq}, \top).\mathit{WSProvider}_5 \\
\mathit{WSProvider}_5 &\stackrel{\mathit{def}}{=} (\mathit{checkValid}, 99 \cdot \top).\mathit{WSProvider}_6 + (\mathit{checkValid}, \top).\mathit{WSProvider}_{10} \\
\mathit{WSProvider}_6 &\stackrel{\mathit{def}}{=} (\mathit{locRes}, r_6).\mathit{WSProvider}_7 \\
\mathit{WSProvider}_7 &\stackrel{\mathit{def}}{=} (\mathit{sendMMS}, \top).\mathit{WSProvider}_8 \\
\mathit{WSProvider}_8 &\stackrel{\mathit{def}}{=} (\mathit{getMap}, r_8).\mathit{WSProvider}_9 \\
\mathit{WSProvider}_9 &\stackrel{\mathit{def}}{=} (\mathit{stopSession}, r_2).\mathit{WSProvider} \\
\mathit{WSProvider}_{10} &\stackrel{\mathit{def}}{=} (\mathit{locErr}, r_6).\mathit{WSProvider}_{11} \\
\mathit{WSProvider}_{11} &\stackrel{\mathit{def}}{=} (\mathit{get404}, r_8).\mathit{WSProvider}_9
\end{aligned}$$

11.4 Component *PAProvider*

In our model the Policy Access Provider has a very simple behaviour. It simply maintains a thread for each session and carries out the validity check on behalf of the Web Service Provider.

$$\begin{array}{rcl} \mathit{PAProvider} & \overset{def}{=} & (\mathit{startSession}, \top).\mathit{PAProvider} \\ & + & (\mathit{checkValid}, r_5).\mathit{PAProvider} \\ & + & (\mathit{stopSession}, \top).\mathit{PAProvider} \end{array}$$

This representation of the *PAProvider* is stateless and we will later contrast its behaviour with an alternative stateful form.

11.5 Model Component *WSComp*

The complete system is represented by some number of instances of the components interacting on their shared activities:

$$\mathit{WSComp} \overset{def}{=} \big((\mathit{Customer}[N_C] \underset{L_1}{\bowtie} \mathit{WSProvider}[N_{WSP}]) \underset{L_2}{\bowtie} \mathit{WSConsumer}[N_{WSC}]\big) \underset{L_3}{\bowtie} \mathit{PAProvider}[N_{PAP}]$$

where the cooperation sets are

$$\begin{array}{rcl} L_1 & = & \{\mathit{getSMS}, \mathit{getMap}, \mathit{get404}\} \\ L_2 & = & \{\mathit{notify}, \mathit{locReq}, \mathit{locRes}, \mathit{locErr}, \mathit{sendMMS}\} \\ L_3 & = & \{\mathit{startSession}, \mathit{checkValid}, \mathit{stopSession}\} \end{array}$$

and N_C, N_{WSC}, N_{WSP} and N_{PAP} are the number of instances of *Customer*, *WSConsumer*, *WSProvider* and *PAProvider* respectively.

11.6 Performance Analysis of the Web Service Composition

In this section we carry out steady-state analysis on the Web Service Composition case study in order to tune the parameters of the system. To accomplish this task we use a modified version of the model in which the customer is explicitly modelled as a component of the system. The values for each rate are shown in Table 3.

Suppose that we want to design the system in such a way that it can handle 30 independent customers. The modeller may have constraints on some parameters such as the network delays because those are limited by the available technology. However, there are a number of degrees of freedom which let her vary, for example, the number of threads of control of the components of the system. The purpose is to deliver a satisfactory service in a cost-effective way. The simplest example of a cost function may be a linearly dependency on the number of copies of a component or the rate at which an activity is performed.

The graph in Fig. 23 shows the throughput of the *getMap* action as the number of customers varies between 1 and 30. Each line represents a given number of copies of the *WSProvider* component in the system. When the total number of customers is 30, two providers lead to a throughput which is twice as much as in the base system configuration

parameter	*value*	*explanation*
r_1	0.0010	rate at which customers request maps
r_2	0.5	rate at which a session can be started
r_3	0.1	notification exchange between consumer and provider
r_4	0.1	rate at which requests for customer's location can be satisfied
r_5	0.05	rate at which the provider can check the validity of the incoming request
r_6	0.1	rate at which location information can be returned to the consumer
r_7	0.05	rate at which maps can be generated
r_8	0.02	rate at which MMS messages can be sent from provider to customer
r_9	$10.0 * r_8$	rate at which MMS messages can be sent via the Web Service

Table 3: Parameters used in the performance analysis of the Web Service composition

with one provider only. However, as the number of providers increases the incremental benefit becomes less significant. In particular, the system with four copies is just 8.7% faster than the system with three. In the following we set N_{WSP}, the number of copies of *WSProvider*, to be 3.

Fig. 24 shows the effect that the rate at which the users initiate the request (r_1) has on the *getMap* throughput for different numbers of copies of the *WSConsumer*. Every line starts to plateau at approximately $r_1 = 0.010$ following an initial sharp increase. This suggests that the system can guarantee satisfactory behaviour under the constraint that the users' request rate is below that threshold. In addition, the graph gives the modeller insights into the suitable number of operating threads of control of *WSConsumer*, which we believe is two as the additional third copy is not well matched by performance boost. Hence, in order to tune *PAProvider*—the remaining system component—we set N_{WSC} to that value.

The same approach can be applied to the optimisation of the number of copies of *PAProvider*. Here we are particularly interested in the overall impact of the rate at which the validity check is performed. Slower rates may mean more computationally expensive validation, whereas faster rates may involve less accuracy and lower security of the system. Such effects are measured in Fig. 25 where the *getMap* throughput is plotted against r_5 for different *PAProvider* pool sizes. A sharp increase followed by a constant levelling off suggests that optimal rate values lie on the left of the plateau, as faster rates do not improve the system considerably. As for the optimal number of copies of *PAProvider*, deploying two copies rather than one dramatically increases the quality of service of the overall system. With a similar approach as previously discussed, the modeller may want to consider the trade-off between the cost of adding a third copy and the throughput increase.

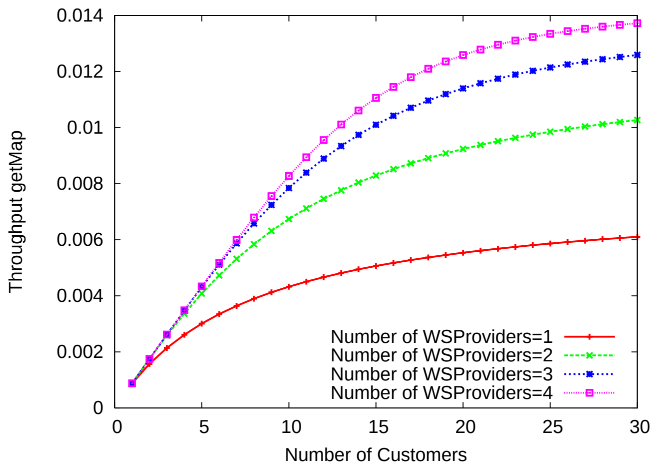

Figure 23: Throughput of *getMap* for varying number of *WSProvider* and customers

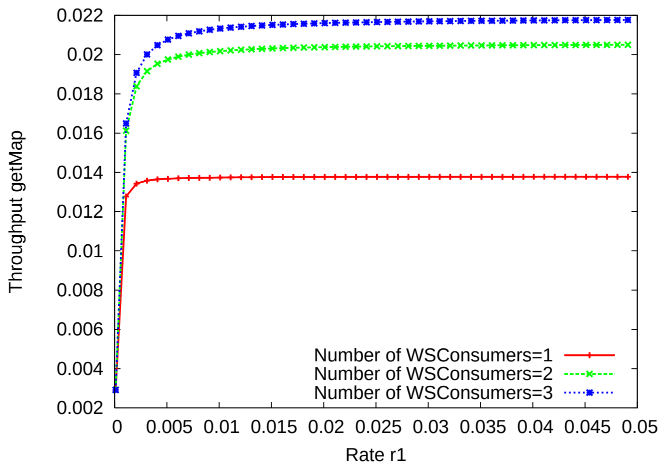

Figure 24: Throughput of *getMap* for varying number of *WSConsumer* and r_1

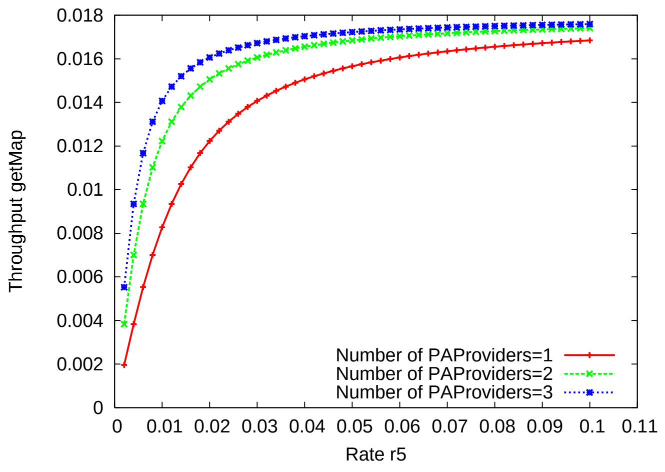

Figure 25: Throughput of *getMap* for varying number of *PAProvider* and r_5

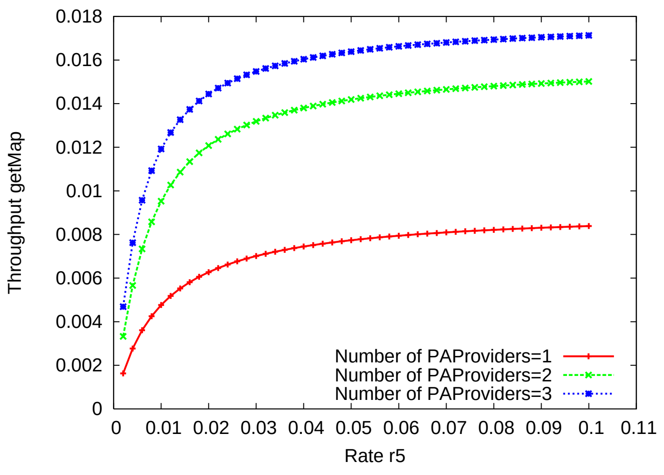

Figure 26: Throughput of *getMap* for varying number of stateful *PAProvider* and r_5

11.7 Evaluation of an alternative design of *PAProvider*

We conclude by showing how this model can be used to evaluate alternative designs of parts of the system. Here, we focus on *PAProvider* which has been originally modelled as a stateless component. Any of its services can be called at any point, the correctness of the system being guaranteed by implementation-specific constraints such as session identifiers being uniquely assigned to the clients and passed as parameters of the method calls.

Another design of a component which offers the same functionalities is that of a stateful provider. In PEPA such a service can be modelled as a sequential component with three local states:

$$\begin{aligned}
PAProvider &\stackrel{def}{=} (startSession, \top).PAProvider_2 \\
PAProvider_2 &\stackrel{def}{=} (checkValid, r_5).PAProvider_3 \\
PAProvider_3 &\stackrel{def}{=} (stopSession, \top).PAProvider
\end{aligned}$$

This implementation has the consequence that there can never at any point be more than N_{WSP} *WSProvider* which have started a session with a *PAProvider*. This is because the provider has to release a previous session in order to start another one.

The graph in Fig. 26 measures the same metrics as in Fig. 25 when the stateful provider is employed. It shows that the incremental gain in adding more copies has become more noteworthy. However, the modeller may prefer the original version, as three copies of the stateful provider deliver about as much throughput as only one copy of the stateless implementation.