Performance Modelling — Lecture 11
PEPA Case Study

Jane Hillston

School of Informatics
The University of Edinburgh
Scotland

Dynamic behaviour

- The behaviour of a model is dictated by the semantic rules which tell us how to interpret the combinators of the language.

Dynamic behaviour

- The behaviour of a model is dictated by the semantic rules which tell us how to interpret the combinators of the language.
- The possible evolutions of a model are captured by applying these rules exhaustively, generating a labelled transition system, or derivation graph.

Dynamic behaviour

- The behaviour of a model is dictated by the semantic rules which tell us how to interpret the combinators of the language.
- The possible evolutions of a model are captured by applying these rules exhaustively, generating a labelled transition system, or derivation graph.
- In this graph, each node is a state of the model (comprised of the local states of each of the components) and the arcs represent the actions causing the move from one state to another.

Dynamic behaviour

- The behaviour of a model is dictated by the semantic rules which tell us how to interpret the combinators of the language.
- The possible evolutions of a model are captured by applying these rules exhaustively, generating a labelled transition system, or derivation graph.
- In this graph, each node is a state of the model (comprised of the local states of each of the components) and the arcs represent the actions causing the move from one state to another.
- This graph can be treated as the state transition diagram of a CTMC, leading to the generation of the infinitesimal generator matrix.

Upgrading a PC LAN

Recall we wish to determine the mean waiting time for data packets at a PC connected to a local area network, operating as a token ring.

Upgrading a PC LAN

Recall we wish to determine the mean waiting time for data packets at a PC connected to a local area network, operating as a token ring.

The transmission medium supports no more than one transmission at any given time. To resolve conflicts, a token is passed round the network from one node to another in round robin order.

Upgrading a PC LAN

Recall we wish to determine the mean waiting time for data packets at a PC connected to a local area network, operating as a token ring.

The transmission medium supports no more than one transmission at any given time. To resolve conflicts, a token is passed round the network from one node to another in round robin order.

A node can transmit, only whilst it holds the token.

Upgrading a PC LAN

There are currently four PCs (or similar devices) connected to the LAN in a small office, but the company has recently recruited two new employees, each of whom will have a PC. Our task is to find out how the delay experienced by data packets at each PC will be affected if another two PCs are added.

Modelling Assumptions

- Each PC can only store one data packet waiting for transmission at a time.

Modelling Assumptions

- Each PC can only store one data packet waiting for transmission at a time.
- When the token arrives either there is one packet waiting or no packet waiting.

Modelling Assumptions

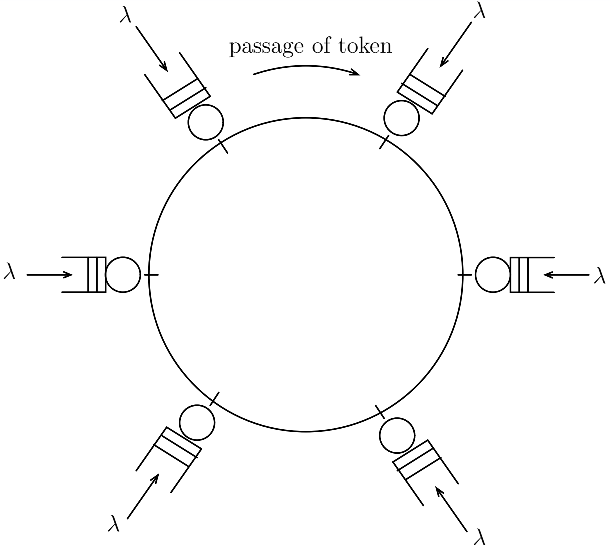
- Each PC can only store one data packet waiting for transmission at a time.
- When the token arrives either there is one packet waiting or no packet waiting.
- The average rate that a PC generates data packets is λ.

Modelling Assumptions

- Each PC can only store one data packet waiting for transmission at a time.
- When the token arrives either there is one packet waiting or no packet waiting.
- The average rate that a PC generates data packets is λ.
- The mean duration of a data packet transmission is d ($d = 1/\mu$), and the mean time for the token to pass from one PC to the next is m ($m = 1/\omega$).

Modelling Assumptions

- Each PC can only store one data packet waiting for transmission at a time.
- When the token arrives either there is one packet waiting or no packet waiting.
- The average rate that a PC generates data packets is λ.
- The mean duration of a data packet transmission is d ($d = 1/\mu$), and the mean time for the token to pass from one PC to the next is m ($m = 1/\omega$).
- Transmission is gated: each PC can transmit at most one data packet per visit of the token.


λ
λ
passage of token
λ
λ
λ
λ

Modelling the system: choosing components

The first stage in developing a model of the system in PEPA is to determine the components of the system and the actions which they can undertake.

Modelling the system: choosing components

The first stage in developing a model of the system in PEPA is to determine the components of the system and the actions which they can undertake.

It seems clear that one type of component should be used to represent the PCs. The components representing the four/six PCs with have essentially the same behaviour. But since token visits the nodes in order we will need to distinguish the components.

Modelling the system: choosing components

The first stage in developing a model of the system in PEPA is to determine the components of the system and the actions which they can undertake.

It seems clear that one type of component should be used to represent the PCs. The components representing the four/six PCs with have essentially the same behaviour. But since token visits the nodes in order we will need to distinguish the components.

We will need another component to represent the medium. As remarked previously, the medium can be represented solely by the token.

Modelling the system: choosing actvities

The description of the PC is very simple in this case. It only has two activities which it can undertake:

- generate a data packet;
- transmit a data packet.

Moreover we are told that it can only hold one data packet at a time and so these activities must be undertaken sequentially.

Modelling the system: choosing actvities

The description of the PC is very simple in this case. It only has two activities which it can undertake:

- generate a data packet;
- transmit a data packet.

Moreover we are told that it can only hold one data packet at a time and so these activities must be undertaken sequentially.

This suggests the following PEPA component for the ith PC:

$$PC_{i0} \stackrel{def}{=} (arrive, \lambda).PC_{i1}$$
$$PC_{i1} \stackrel{def}{=} (transmit_i, \mu).PC_{i0}$$

Modelling the system: choosing actvities

The description of the PC is very simple in this case. It only has two activities which it can undertake:

- generate a data packet;
- transmit a data packet.

Moreover we are told that it can only hold one data packet at a time and so these activities must be undertaken sequentially.

This suggests the following PEPA component for the ith PC:

$$
\begin{aligned}
PC_{i0} &\stackrel{def}{=} (arrive, \lambda).PC_{i1} \\
PC_{i1} &\stackrel{def}{=} (transmit_i, \mu).PC_{i0}
\end{aligned}
$$

This will need some refinement when we consider interaction with the token.

Modelling the system: choosing activities

For the token we can think of its current state being characterised by its current position. Thus, if there are N PCs in the network the states of the token correspond to the values $\{1, 2, \ldots N\}$.

Modelling the system: choosing activities

For the token we can think of its current state being characterised by its current position. Thus, if there are N PCs in the network the states of the token correspond to the values $\{1, 2, \ldots N\}$.

When it is at the ith PC then the token may

- transmit a token if there is one to transmit and then walk on; or
- walk on at once if there is no token waiting.

Modelling the system: choosing activities

For the token we can think of its current state being characterised by its current position. Thus, if there are N PCs in the network the states of the token correspond to the values $\{1, 2, \ldots N\}$.

When it is at the ith PC then the token may

- transmit a token if there is one to transmit and then walk on; or
- walk on at once if there is no token waiting.

$$\mathit{Token}_i \stackrel{\mathit{def}}{=} (\mathit{walkon}_{i+1}, \omega).\mathit{Token}_{i+1} + (\mathit{transmit}_i, \mu).(\mathit{walk}_{i+1}, \omega).\mathit{Token}_{i+1}$$

Refining the components

In order to ensure that the token's choice is made dependent on the state of PC being visited, we add a walkon action to the PC when it is empty, and impose a cooperation between the PC and the Token for both walkon and serve.

Refining the components

In order to ensure that the token's choice is made dependent on the state of PC being visited, we add a walkon action to the PC when it is empty, and impose a cooperation between the PC and the Token for both walkon and serve.

$$PC_{i0} \stackrel{def}{=} (arrive, \lambda).PC_{i1} + (walkon_2, \omega).PC_{i0}$$
$$PC_{i1} \stackrel{def}{=} (transmit_i, \mu).PC_{i0}$$

Complete model: four PC case

$$PC_{10} \stackrel{def}{=} (arrive, \lambda).PC_{11} + (walkon_2, \omega).PC_{10}$$
$$PC_{11} \stackrel{def}{=} (transmit_1, \mu).PC_{10}$$

$$PC_{20} \stackrel{def}{=} (arrive, \lambda).PC_{21} + (walkon_3, \omega).PC_{20}$$
$$PC_{21} \stackrel{def}{=} (transmit_2, \mu).PC_{20}$$

$$PC_{30} \stackrel{def}{=} (arrive, \lambda).PC_{31} + (walkon_4, \omega).PC_{30}$$
$$PC_{31} \stackrel{def}{=} (transmit_3, \mu).PC_{30}$$

$$PC_{40} \stackrel{def}{=} (arrive, \lambda).PC_{41} + (walkon_1, \omega).PC_{40}$$
$$PC_{41} \stackrel{def}{=} (transmit_4, \mu).PC_{40}$$

$$Token_1 \stackrel{def}{=} (walkon_2, \omega).Token_2 + (transmit_1, \mu).(walk_2, \omega).Token_2$$
$$Token_2 \stackrel{def}{=} (walkon_3, \omega).Token_3 + (transmit_2, \mu).(walk_3, \omega).Token_3$$
$$Token_3 \stackrel{def}{=} (walkon_4, \omega).Token_4 + (transmit_3, \mu).(walk_4, \omega).Token_4$$
$$Token_4 \stackrel{def}{=} (walkon_1, \omega).Token_1 + (transmit_4, \mu).(walk_1, \omega).Token_1$$

$$LAN \stackrel{def}{=} (PC_{10} \parallel PC_{20} \parallel PC_{30} \parallel PC_{40}) \bowtie_{L} Token_1$$
$$\text{where } L = \{walkon_1, walkon_2, walkon_3, walkon_4, serve_1, serve_2, serve_3, serve_4\}.$$

Here we have arbitrarily chosen a starting state in which all the PCs are empty and the Token is at PC1.

Web Service Composition: Introduction

We consider an example of a business application which is composed from a number of offered web services.

Web Service Composition: Introduction

We consider an example of a business application which is composed from a number of offered web services.

A user accesses the application via an SMS message requesting directions to the nearest facility (post-office, restaurant, bank etc.) and receives a response as an MMS message containing a map.

Web Service Composition: Introduction

We consider an example of a business application which is composed from a number of offered web services.

A user accesses the application via an SMS message requesting directions to the nearest facility (post-office, restaurant, bank etc.) and receives a response as an MMS message containing a map.

Since the application involves a users' current location there is an access control issue since it must be ensured that the web service consumer has the requisite authority to execute the web service it requests.

Web Service Composition: Introduction

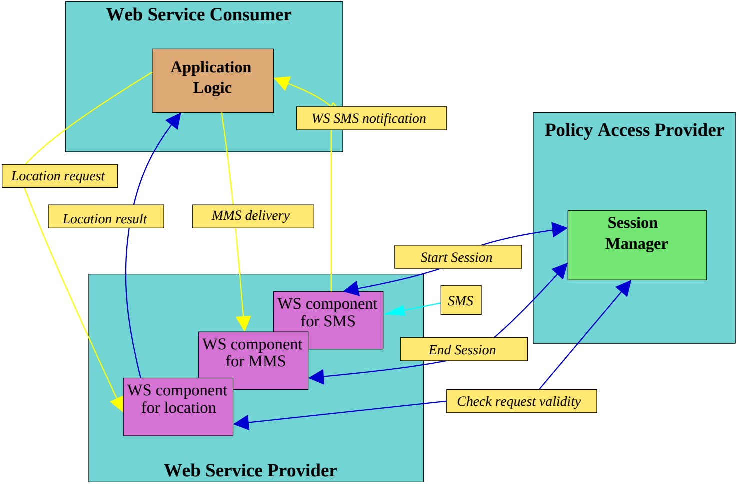
We consider an example of a business application which is composed from a number of offered web services.

A user accesses the application via an SMS message requesting directions to the nearest facility (post-office, restaurant, bank etc.) and receives a response as an MMS message containing a map.

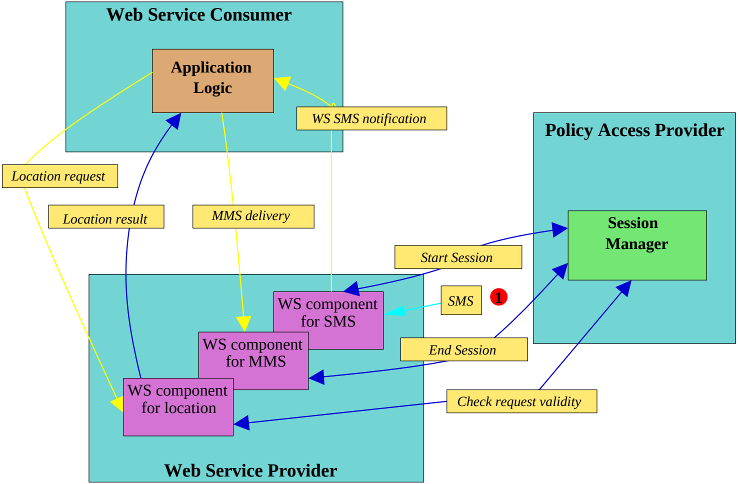
Since the application involves a users' current location there is an access control issue since it must be ensured that the web service consumer has the requisite authority to execute the web service it requests.

Moreover the service provider imposes a restriction that only one request may be handled for each SMS message received.

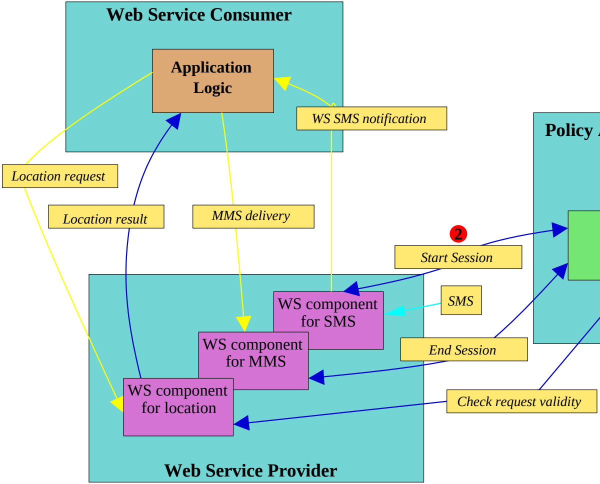
Schematic view



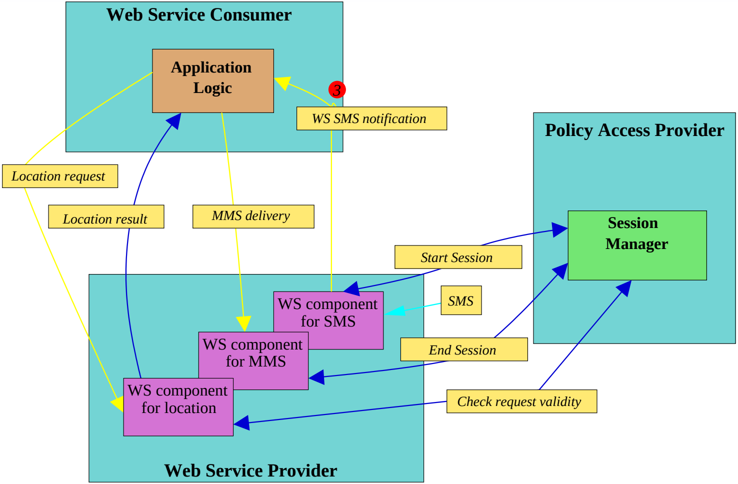
Schematic view



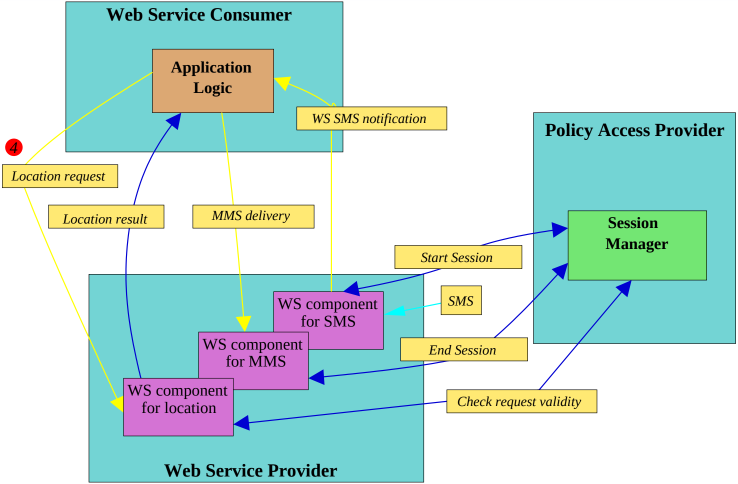
Schematic view



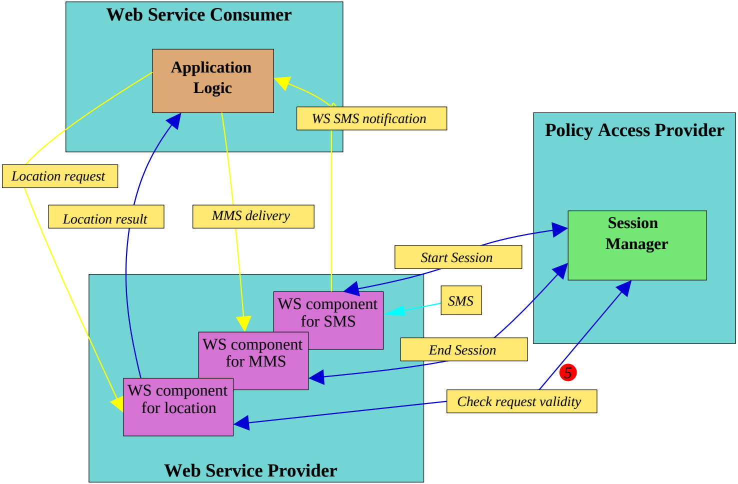
Schematic view



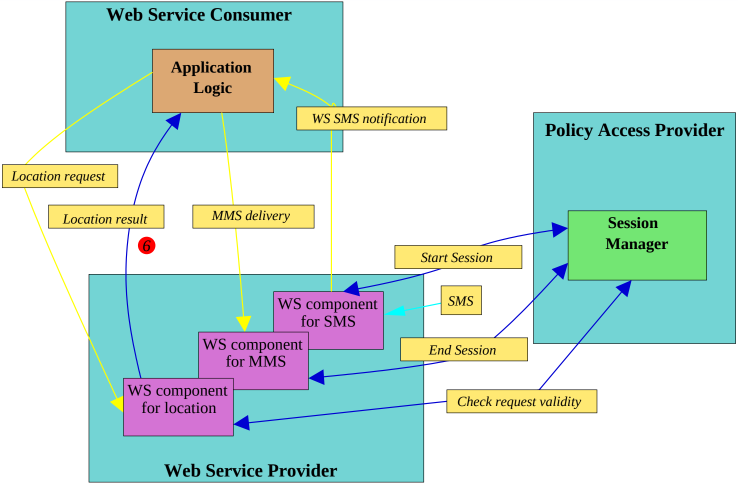
Schematic view



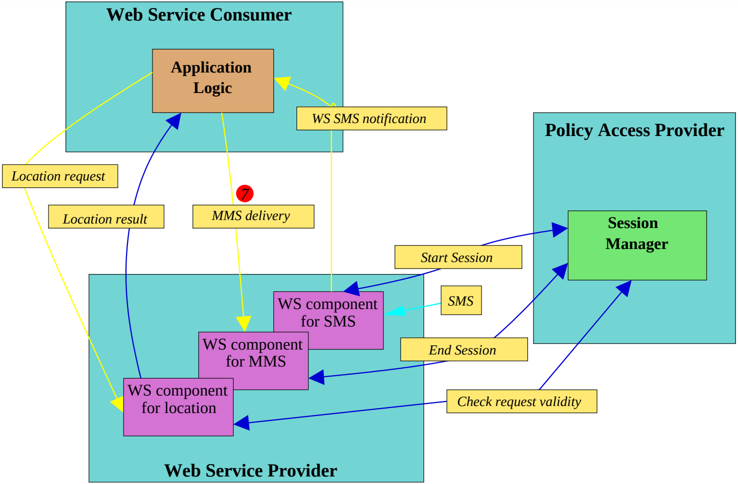
Schematic view



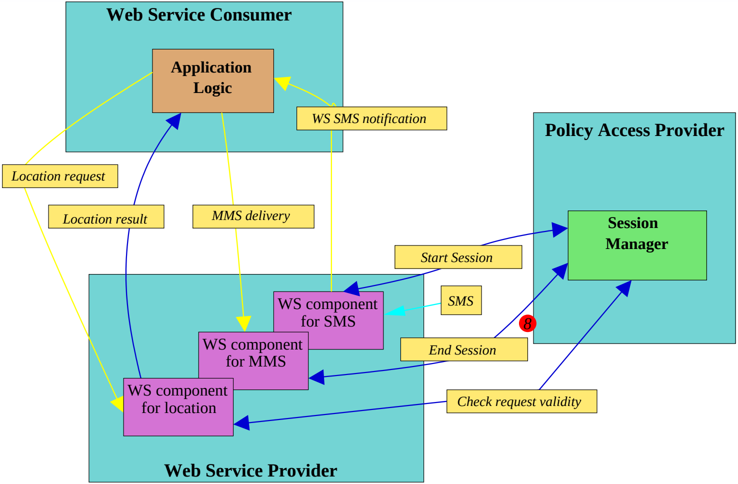
Schematic view



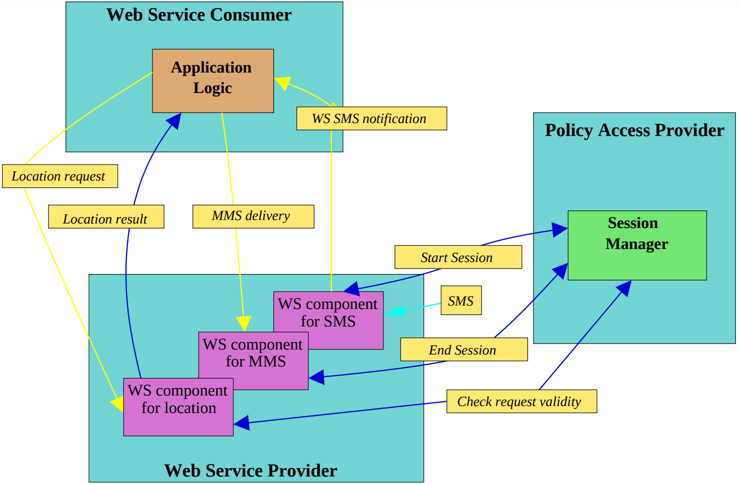
Schematic view



Schematic view



Schematic view



The PEPA model

The PEPA model of the system consists of four components:

The PEPA model

The PEPA model of the system consists of four components:

- The user;
- The web service provider;
- The web service consumer, and
- The policy access provider.

The PEPA model

The PEPA model of the system consists of four components:

- The user;
- The web service provider;
- The web service consumer, and
- The policy access provider.

The Web Service Provider consists of three distinct elements but the web service consumer is associated with a session which accesses each element in sequence.

The PEPA model

The PEPA model of the system consists of four components:

- The user;
- The web service provider;
- The web service consumer, and
- The policy access provider.

The Web Service Provider consists of three distinct elements but the web service consumer is associated with a session which accesses each element in sequence.

Concurrency is introduced into the model by allowing multiple sessions rather than by representing the constituent web services separately.

Component *Customer*

The customer's behaviour is simply modelled with two local states.

Component *Customer*

The customer's behaviour is simply modelled with two local states.

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

Component *Customer*

The customer's behaviour is simply modelled with two local states.

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

We associate the user-perceived system performance with the throughput of the *getMap* action which can be calculated directly from the steady state probability distribution of the underlying Markov chain.

Component *WSConsumer*

Once a session has been started, it initiates a request for the user's current location and waits for a response.

Component *WSConsumer*

Once a session has been started, it initiates a request for the user's current location and waits for a response.
For valid requests, location is returned and used to compute the appropriate map, which is then sent via an MMS message, using the web service.

Component *WSConsumer*

Once a session has been started, it initiates a request for the user's current location and waits for a response.
For valid requests, location is returned and used to compute the appropriate map, which is then sent via an MMS message, using the web service.

$$
\begin{aligned}
WSConsumer &\stackrel{def}{=} (notify, \top).WSConsumer_2 \\
WSConsumer_2 &\stackrel{def}{=} (locReq, r_4).WSConsumer_3 \\
WSConsumer_3 &\stackrel{def}{=} (locRes, \top).WSConsumer_4 \\
&\;\;+\;\; (locErr, \top).WSConsumer \\
WSConsumer_4 &\stackrel{def}{=} (compute, r_7).WSConsumer_5 \\
WSConsumer_5 &\stackrel{def}{=} (sendMMS, r_9).WSConsumer
\end{aligned}
$$

Component *WSProvider*

The use of sessions restricts a user's access to the services of the Web Service Provider to be sequential.

Component *WSProvider*

The use of sessions restricts a user's access to the services of the Web Service Provider to be sequential.

We assume that there is a distinct instance of the component *WSProvider* for each distinct session.

Component *WSProvider*

The use of sessions restricts a user's access to the services of the Web Service Provider to be sequential.

We assume that there is a distinct instance of the component *WSProvider* for each distinct session.

The *checkValid* action is represented twice, to capture the two possible distinct outcomes of the action.

Component *WSProvider*

The use of sessions restricts a user's access to the services of the Web Service Provider to be sequential.

We assume that there is a distinct instance of the component *WSProvider* for each distinct session.

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*).

Component *WSProvider*

The use of sessions restricts a user's access to the services of the Web Service Provider to be sequential.

We assume that there is a distinct instance of the component *WSProvider* for each distinct session.

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*).
- 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*).

Component *WSProvider*

$$
\begin{aligned}
WSProvider &\stackrel{def}{=} (getSMS, \top).WSProvider_2 \\
WSProvider_2 &\stackrel{def}{=} (startSession, r_2).WSProvider_3 \\
WSProvider_3 &\stackrel{def}{=} (notify, r_3).WSProvider_4 \\
WSProvider_4 &\stackrel{def}{=} (locReq, \top).WSProvider_5 \\
WSProvider_5 &\stackrel{def}{=} (checkValid, 99 \cdot \top).WSProvider_6 \\
&+ (checkValid, \top).WSProvider_{10}
\end{aligned}
$$

Component *WSProvider* cont.

$$
\begin{aligned}
WSProvider_6 &\stackrel{def}{=} (locRes, r_6).WSProvider_7 \\
WSProvider_7 &\stackrel{def}{=} (sendMMS, \top).WSProvider_8 \\
WSProvider_8 &\stackrel{def}{=} (getMap, r_8).WSProvider_9 \\
WSProvider_9 &\stackrel{def}{=} (stopSession, r_2).WSProvider \\
WSProvider_{10} &\stackrel{def}{=} (locErr, r_6).WSProvider_{11} \\
WSProvider_{11} &\stackrel{def}{=} (get404, r_8).WSProvider_9
\end{aligned}
$$

Component *PAProvider*

We consider a stateless implementation of the policy access provider.

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

Model Component *WSComp*

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

$$
\begin{aligned}
WSComp \stackrel{def}{=} & ((Customer[N_C] \underset{L_1}{\bowtie} WSProvider[N_{WSP}]) \\
& \underset{L_2}{\bowtie} WSConsumer[N_{WSC}]) \\
& \underset{L_3}{\bowtie} PAProvider[N_{PAP}]
\end{aligned}
$$

where the cooperation sets are

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

Parameter Values

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

Steady State Analysis for System Tuning

- Suppose that we want to design the system in such a way that it can handle 30 independent customers.

Steady State Analysis for System Tuning

- Suppose that we want to design the system in such a way that it can handle 30 independent customers.
- Some parameters such as the network delays may be constrained by the available technology.

Steady State Analysis for System Tuning

- Suppose that we want to design the system in such a way that it can handle 30 independent customers.
- Some parameters such as the network delays may be constrained by the available technology.
- However, there are a number of degrees of freedom which let us vary, for example, the number of threads of control of the components of the system.

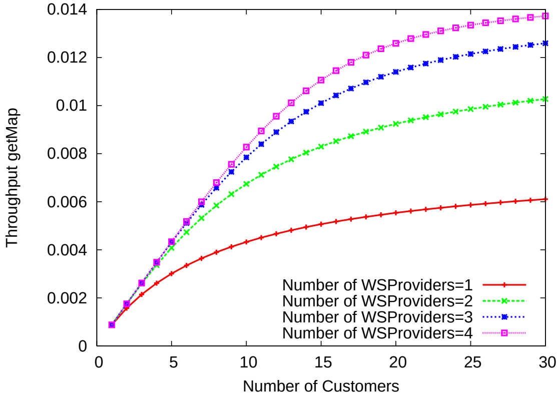
Steady State Analysis for System Tuning

- Suppose that we want to design the system in such a way that it can handle 30 independent customers.
- Some parameters such as the network delays may be constrained by the available technology.
- However, there are a number of degrees of freedom which let us vary, for example, the number of threads of control of the components of the system.
- The aim of the analysis is to deliver a satisfactory service in a cost-effective way.

Steady State Analysis for System Tuning

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

Throughput of the *getMap* action

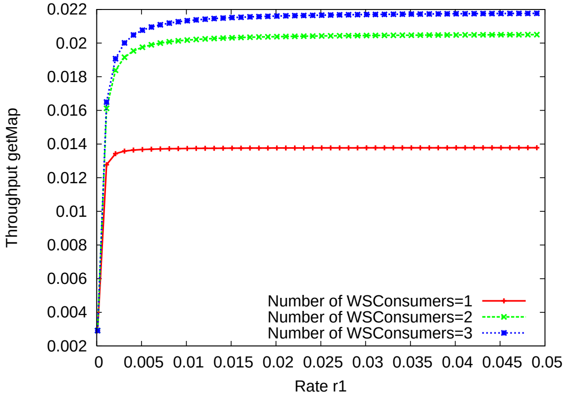


as the number of customers varies between 1 and 30 for various numbers of copies of the *WSProvider* component.

Throughput of the *getMap* action

- Under heavy load increasing the number of providers initially leads to a sharp increase in the throughput. However the gain deteriorates so that the system with four copies is just 8.7% faster than the system with three.
- In the following we settle on three copies of *WSProvider*.

Throughput of *getMap* action



as the request arrival rate (r_1) varies for differing numbers of *WSConsumer*.

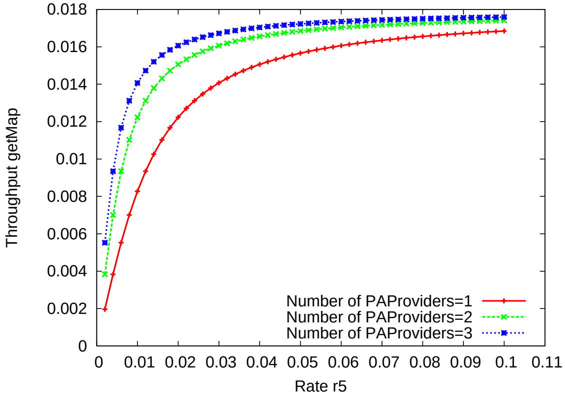
Throughput of *getMap* action

- Every line starts to plateau at approximately $r_1 = 0.010$ following an initial sharp increase. This suggests that the user is the bottleneck in the system when the arrival rate is lower. Conversely, at high rates the system becomes congested.
- Whilst having two copies of *WSConsumer*, corresponding to two operating threads of control, improves performance significantly, the subsequent increase with three copies is less pronounced.
- So we set the number of copies of *WSConsumer* to 2.

Optimising 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.
- Faster rates may involve less accuracy and lower security of the system.

Throughput of *getMap* action



as the validity check rate (r_5) varies for differing numbers of *PAProvider*.

Throughput of *getMap* action

- 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.

An alternative design for *PAProvider*

- The original design of *PAProvider* is stateless.
- 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.

An alternative design for *PAProvider*

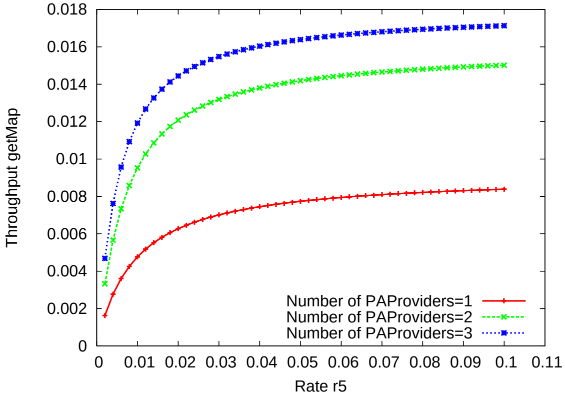
- The original design of *PAProvider* is stateless.
- Any of its services can be called at any point, the correctness of the system being guaranteed by implementation-specific constraints such as session identifiers being uniquely assigned to the clients and passed as parameters of the method calls.
- Alternatively we may consider a stateful implementation, modelled as a sequential component with three local states.
- This implementation has the consequence that there can never be more than N_{PAP} *WSProvider* which have started a session with a *PAProvider*

Component *PAProvider* — Stateful Version

It maintains a thread for each session and carries out the validity check on behalf of the Web Service Provider.

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

Throughput of *getMap* action



as the validity check rate (r_5) varies for differing numbers of *PAProvider* (stateful version).

Throughput of *getMap* action

- In this case the incremental gain in adding more copies has become more marked.
- However, the modeller may want to prefer the original version, as three copies of the stateful provider deliver about as much as the throughput of only one copy of the stateless implementation.

Acknowledgement

Modelling the web service composition system was joint work with Mirco Tribastone.