Calculating Web Service Interaction Using Extended BPEL Language

Csaba Legány, Márk Kaszó, Tihamér Levendovszky

Budapest University of Technology and Economics, Hungary
legcsabi@aut.bme.hu, mkaszo@aut.bme.hu, tihamer@aut.bme.hu

Abstract: Nowadays web services and SOA (Service Oriented Architecture) systems have fundamental significance in computer science, especially in platform integration. The most common consideration is system workload estimation. Interacting composite web services can be modeled as asynchronously interacting BPEL processes. In order to compute key system workload parameters in design time – such as service time of a process, response time or resource cost – the standard BPEL model was extended with new terms and attributes. This paper formalizes the computation of these parameters using a recursive algorithm. Finally a numerical example is presented.

Keywords: BPEL, extension, web services, modeling, SOA, XML

1 Introduction

Business Process Execution Language [1] gives us a standard way to specify our business processes. This paper deals with problem of performance prediction in SOA systems in design time. BPEL and XML [5] are suitable to describe SOA systems, but the original version of BPEL is unable to describe performance parameters. A new BPEL extension is presented in Modeling Extended BPEL Language [2]. This paper deals with the calculation of system workload parameters of interacting web services [4] in design time. Key system workload parameters such as service time, response time and resource cost can be calculated using the BPEL extension.

The structure of the article is as follows: In Section 2 the key factors of system workload are calculated using mathematical formalism. There is a complex numerical example in Section 3. Finally Section 4 summarizes our work.

2 Calculations based on BPEL Extension

There are several system workload factors that can be calculated even in design time using the BPEL extension described in the article Modeling extended BPEL language [2]. The most important workload characteristics of a system are the following:

From the aspect of the server [3]:

➔ Throughput : amount of finished requests/sec

➔ Resource utilization: measures the usage of resources (CPU, Memory, Disk IO, Network)

➔ Service time of a selected process: the time required for a process to finish its execution and return its return value.

➔ Resource cost of a selected process: the cost (required memory, IO, CPU) of a process on a server.

From the aspect of the client:

➔ Response time: the amount of time required to get response from the server for a given request. It can be either the time to get the last byte of the server response (TTLB) or to get the first byte of the response (TTFB). Here we will calculate in case of TTFB.

2.1 Service Time and Response Time Calculation

Service time is the time required for a process to return its return value. This section will focus on the calculation of service time using other parameters like serialization time, link time and data transformation time. For example if a new user registers on a web site, we can have three processes:

- P_{web} running on the web server
- P_{app} running on the application server
- P_{DB1} and P_{DB2} (two sub-processes) running on the database server.

P_{app} can only return if P_{DB1} and P_{DB2} have already returned (probably P_{DB1} and P_{DB2} can run parallel), P_{web} can only return if P_{app} has already returned. A graph displaying interacting processes is called as process interaction graph (Figure 2).

Let us denote server *S* of process *P* with $P.S$. It means that *P* runs on *S*. A process is P-sending if it sends messages to P, similarly a process is P-receiving if it receives messages from P. Let us denote messages with *M*. $M(P_1, P_2)$ stands for a message sent by process P_1 to P_2 which has to be serialized and deserialized on the server of P_1, P_2, (formally on $P_2.S$, $P_1.S$). Let us define link time $T_{link}(P, P_i)$ as the average delay of a link between processes P and P_i: $T_{link}(P, P_i) = link(P, P_i).avg\_delay$.

Let us define serialization time, $T_{ser}(P_i, P)$ as:

$T_{ser}(P_i, P) = \sum_{M_j \in M(P_i, P)} T_{ser}(M_j(P_i, P), P.S)$ where

- $M(P_i, P)$ is the set of all messages sent by P_i to P, M_j is a selected message of this set
- $T_{ser}(M_j(P_i, P), P.S)$ is the serialization cost of the selected message on the server of the receiver (formally on $P.S$). Section 3 details the computation of this value.

Let us suppose that waiting delay of any process *P* is constant, $T_{wait}(P) = const_1$ In a more advanced model using workload estimation $T_{wait}(P)$ will not be constant. Every process converts its input messages to output messages (this is usually value transformation). Let us denote the time of this transformation with $T_{data\_transform}(P)$ for process *P*. Let $T_{data\_transform}(P) = const_2$. In a more advanced model it can depend on the types of data to convert. However, according to our measurements, data transformation time is much smaller than serialization time, it is $O(0)$. Let us define the service time of a process *P*, $T_{serv}(P)$ as the following: if the process receives its first input message at time t_0 and sends its last reply message at t_1, than $T_{serv}(P) = t_1 - t_0$. It is important to note that if a process P interacts with process P_i,than $T_{serv}(P) \geq T_{serv}(P_i)$. Figure 1 depicts such a case for process P interacting with process P_i.

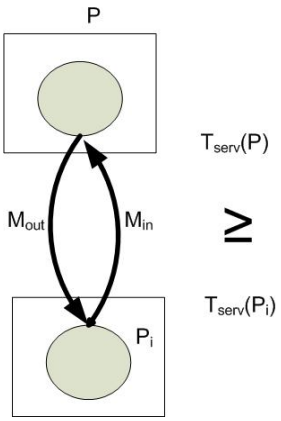

Figure 1
Process interaction graph

Figure 2 illustrates the process interaction graph of a system running on three different servers (e.g. web server, application server, DB server).

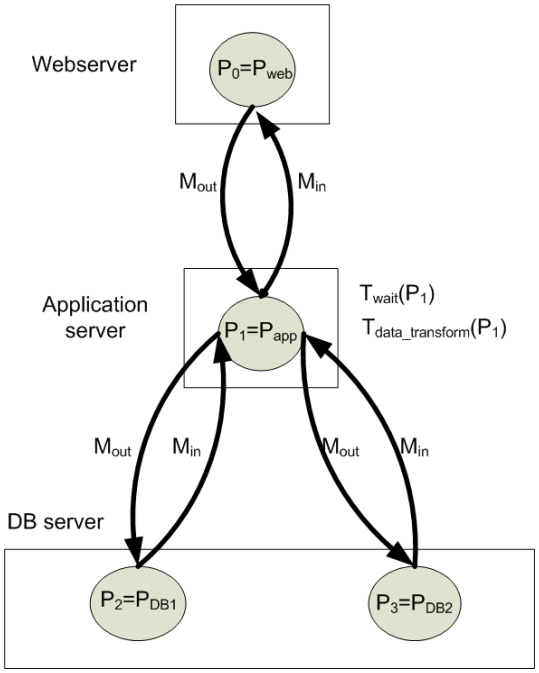

Figure 2
Process interaction graph of a three-tier architecture

$T_{serv}(P)$ can be calculated using the previously defined values.

Let $T_{inner}(P) = T_{wait}(P) + T_{data\_transform}(P)$.

$T_{serv}(P) = T_{inner}(P) + T_{interact}(P)$ where $T_{interact}(P)$ is the summary of service times for all processes *P* interacts with.

Interaction time depends on input and output interaction:

$$T_{interact}(P) = T_{interact\_IN}(P) + T_{interact\_OUT}(P)$$

Both $T_{interact\_IN}(P)$ and $T_{interact\_OUT}(P)$ depend on the interaction method of *P*. Let us focus now only on input interaction.

- If *P* interacts with all of its input processes sequentially: $T_{interact\_IN}(P) = \sum_{P_i \in P_{in}(P)} T_{ser}(P_i, P) + T_{link}(P_i, P) + T_{inner}(P_i)$ where $P_{in}(P)$ is the set of P-sending processes.
- If *P* interacts with all of its input processes parallel: $T_{interact\_IN}(P) = \max_{P_i \in P_{in}(P)} \{T_{ser}(P_i, P) + T_{link}(P_i, P) + T_{inner}(P_i)\}$ which means that the input interaction time of P is determined by the maximum input serialization time.
- Generally,

$$T_{interact\_IN}(P) = \sum_{P_i \in P_{in}^{seq}(P)} T_{ser}(P_i, P) + T_{link}(P_i, P) + T_{inner}(P_i) +$$

$$+ \max_{P_j \in P_{in}^{par}(P)} \{T_{ser}(P_j, P) + T_{link}(P_j, P) + T_{inner}(P_j)\}$$ where

$P_{in}^{seq}(P) \subseteq P_{in}(P)$ is the set of sequential P-sending processes and $P_{in}^{par}(P) \subseteq P_{in}(P)$ is the set of parallel P-sending processes. Similarly,

$$T_{interact\_OUT}(P) = \sum_{P_i \in P_{out}^{seq}(P)} T_{ser}(P_i, P) + T_{inner}(P_i) +$$

$$\max_{P_j \in P_{out}^{par}(P)} \{T_{ser}(P_j, P) + T_{inner}(P_j)\}$$

One can now calculate the service time of any process with these recursive equations. Response time (the time needed to get response from the server for a request) is the service time of the top-level process of the process interaction

graph ($T_{resp} = T_{serv}(P_0)$). For example $T_{resp}(register) = T_{serv}(P_{webregister})$ (where $P_{webregister} \in P_{web}$) in the web user registration example.

Figure 3 displays an example for this recursive approach. The message sent from P to P_i will arrive to P_i in $T_{link}(P_i)$ time. P_i has to wait T_{wait} time before it can start processing the input message. After serialization, data has to be transformed and serialized again. Finally P_i sends its reply back to P over the network. T_{DT} denotes $T_{data\_transform}$ in the figure.

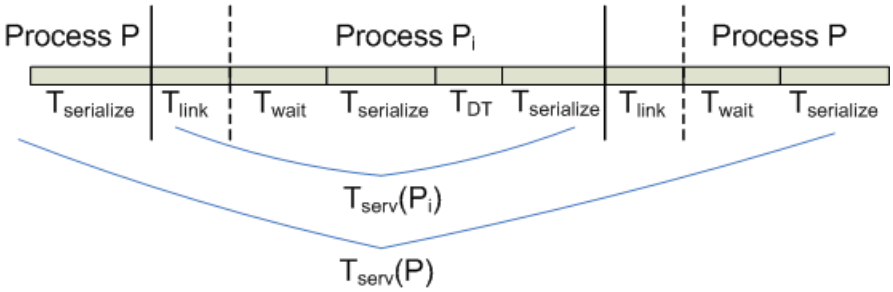

Figure 3
Recursive service time approach

2.2 Resource Cost Calculation

This section will focus on the calculation of resource cost of processes. Every process determines the server it is running on, and is made up of sub-processes. Let us denote all types of cost with L, cost of a given *M* message on server *S* can be obtained as:

$$L(M,S) = \sum_{t \in M.types} S.cost(t) * M.unit \quad \text{where}$$

- *t* is a selected type of message *M*
- $S.cost(t)$ is the cost of *t* on server *S*
- $M.unit$ denotes the number of units for type *t* in message *M*

Note that $S.cost(t)$ can be either memory, IO or CPU cost. Different types of costs (e.g. memory and IO) and costs on different servers (e.g. CPU cost on application server and web server) should not be summarized.

Each sub-process has multiple input and output messages, thus the cost of a sub-process P_{sub1} can be calculated as:

$L(P_{sub1}) = \sum_{\forall M_{in}} L(M_{in}, P_{sub1}.S) + \sum_{\forall M_{out}} L(M_{out}, P_{sub1}.S)$ where

- $\sum_{\forall M_{in}} L(M_{in}, P_{sub1}.S)$ is the summary cost of all input messages of P_{sub1} on server *S*.
- $\sum_{\forall M_{out}} L(M_{out}, P_{sub1}.S)$ is the summary cost of all output messages of P_{sub1} on server *S*.

The cost of each process is the summary cost of its sub-processes: $L(P) = \sum_{\forall i \in P_{sub}(P)} L(P_i)$ where $P_{sub}(P)$ is the set of sub-processes of *P*.

3 Example

Let us suppose that a new user registers on a web site and finally P_{DB1} and P_{DB2} sub-processes are two stored procedures (processes) running on the database server, P_{DB1} stores the user's data while P_{DB2} is used for logging purposes. The input message of P_{DB1} (UserInfo) and the input message of P_{DB2} (UserRegistration) can be found in Listing 1. Both processes will reply with a message containing one boolean variable.

```
<message name="UserInfo">
        <part name="UserName" type="xsd:string" size=”10” unit=”1”/>
        <part name="Password" type="xsd:string" size=”20” unit=”2”/>
</message>
<message name=”UserRegistration”>
        <part name=”date” type=”xsd:date” size=”10” unit=”1”>
</message>
```

Listing 1
Example input messages of P_{DB1} and P_{DB2}

3.1 Calculating Serialization Costs on the Database Server

Now we will focus only on serialization costs. P_{DB1} and P_{DB2} are running on the same S_DB database server, which has the following serialization costs displayed in Listing 2.

```
<server name="S_DB">
        <type name="xsd:string">
        <cost name="serialization" value="10" /></type>
        <type name="xsd:date">
        <cost name="serialization" value="5" /></type>
        <type name="xsd:boolean">
        <cost name="serialization" value="1" /></type>
</server>…
```

Listing 2
Example serialization costs

The total serialization time of P_{DB1} is

$$T_{ser}(P_{DB1}) = T_{ser}(M(P_{app}, P_{DB1}), P_{DB1}.S) + T_{ser}(M(P_{DB1}, P_{app}), P_{app}.S) =$$
$$(10*1+20*2)*10+1*1*1=501$$

Similarly, $T_{ser}(P_{DB2}) = 10*1*5+1*1*1+0=51$

The total serialization time on the database server is 552.

3.3 Calculating Service Times

Let us suppose that

$$T_{data\_transform}(P_{DB1}) = 100\ ,\ T_{data\_transform}(P_{DB2}) = 20\ ,$$
$$T_{data\_transform}(P_{app}) = 100$$

$$T_{wait}(P_{DB1}) = 10\ ,\ T_{wait}(P_{DB2}) = 10\ ,\ T_{wait}(P_{app}) = 10$$

$$T_{link}(P_{app}, P_{DB1}) = T_{link}(P_{DB1}, P_{app}) = T_{link}(P_{app}, P_{DB2}) =$$
$$= T_{link}(P_{DB2}, P_{app}) = 10$$
$$T_{link}(P_{app}, P_{web}) = T_{link}(P_{web}, P_{app}) = 10$$

It means that

$T_{inner}(P_{DB1}) = T_{data\_transform}(P_{DB1}) + T_{wait}(P_{DB1}) = 110$ and
$T_{inner}(P_{DB2}) = 20 + 10 = 30$
$T_{serv}(P_{DB1}) = T_{wait}(P_{DB1}) + T_{data\_transform}(P_{DB1}) + T_{interact}(P_{DB1}) =$
$= 110 + T_{interact}(P_{DB1})$ Similarly, $T_{serv}(P_{DB2}) = 30 + T_{interact}(P_{DB2})$

Interaction time is based on message and link times. Both database sub-processes interact only with P_{app}.

$T_{interact}(P_{DB1}) = T_{interact\_IN}(P_{DB1}) + T_{interact\_OUT}(P_{DB1}) = T_{ser}(P_{app}, P_{DB1}) +$
$+T_{link}(P_{app}, P_{DB1}) + T_{ser}(P_{DB1}, P_{app})$
$T_{ser}(P_{app}, P_{DB1}) = T_{ser}(M(P_{app}, P_{DB1}), P_{DB1}.S) = 500$
$T_{ser}(P_{DB1}, P_{app}) = T_{ser}(M(P_{DB1}, P_{app}), P_{DB1}.S) = 1$

thus $T_{interact}(P_{DB1}) = 500 + 10 + 1 = 511$.

Similarly $T_{interact}(P_{DB2}) = 60 + 10 + 1 = 71$.

Finally we get:

$T_{serv}(P_{DB1}) = 110 + 511 = 621$ and $T_{serv}(P_{DB2}) = 30 + 71 = 101$

We can also compute the service time of P_{app} as follows. Let us suppose that serialization costs are double on the application server than on the database server according to Listing 3.

```
<server name=”S_APP”>
        <type name=”xsd:string”>
        <cost name=”serialization” value=”20” /></type>
        <type name=”xsd:date”>
        <cost name=”serialization” value=”10” /></type>
        <type name=”xsd:boolean”>
        <cost name=”serialization” value=”2” /></type>
</server>
```

Listing 3
Serialization costs on the application server

Let us suppose that P_{app} receives the following input message (Listing 4) from P_{web}. It is important to note that the input message of P_{app} does not contain a UserRegistration input message for P_{DB2}, P_{app} knows that if it receives a UserInfo

message, it should send a UserInfo message to P_{DB1} and a UserRegistration message to P_{DB2}.

```
<message name="UserInfo">
        <part name="UserName" type="xsd:string" size=”10” unit=”1”/>
        <part name="Password" type="xsd:string" size=”20” unit=”2”/>
</message>
```

Listing 4
Input message of P_{app} from P_{web}

The serialization time of this input message is:

$$T_{ser}(P_{web}, P_{app}) = T_{ser}(M(P_{web}, P_{app}), P_{app}.S) = (10*1+20*2)*20$$

P_{app} sends two messages two the database sub-processes, as shown in Figure 2. Their serialization time is:

$$T_{ser}(P_{app}, P_{DB1}) = (10*1+20*2)*20 = 1000$$

$$T_{ser}(P_{app}, P_{DB2}) = 10*1*10 = 100$$

P_{app} receives two messages from the database sub-processes, the serialization time of the response of P_{DB1} is: $T_{ser}(P_{DB1}, P_{app}) = 2*1*1 = 2$. Similarly, $T_{ser}(P_{DB2}, P_{app}) = 2$.

P_{app} sends one message containing a boolean variable to P_{web}. The serialization time is: $T_{ser}(P_{app}, P_{web}) = 1*1*2 = 2$

The interaction time of P_{app} is:

$$T_{interact}(P_{app}) = T_{interact\_IN}(P_{app}) + T_{interact\_OUT}(P_{app})$$

$$T_{interact\_IN}(P_{app}) = T_{ser}(P_{web}, P_{app}) + T_{link}(P_{web}, P_{app}) +$$

$$\max\begin{Bmatrix} T_{ser}(P_{DB1}, P_{app}) + T_{link}(P_{DB1}, P_{app}) + T_{inner}(P_{DB1}) \\ T_{ser}(P_{DB2}, P_{app}) + T_{link}(P_{DB2}, P_{app}) + T_{inner}(P_{DB2}) \end{Bmatrix} =$$

$$= 1000 + 10 + \max\{122, 42\} = 1132$$

since P_{app} receives $M(P_{app}, P_{DB1})$ and $M(P_{app}, P_{DB2})$ from two parallel-running processes.

$T_{interact\_OUT}(P_{app}) = T_{ser}(P_{app}, P_{web}) + T_{ser}(P_{app}, P_{DB1}) + T_{ser}(P_{app}, P_{DB2}) =$
$= 2 + 1000 + 100 = 1102$. Now we can compute $T_{serv}(P_{app})$ as:

$T_{serv}(P_{app}) = T_{wait}(P_{app}) + T_{data\_transform}(P_{app}) + T_{serv}^{interact}(P_{app}) =$
$= 110 + 1132 + 1102 = 2344$ (msec)

Conclusions and Future Works

This paper introduced a method to calculate key parameters in a web service interaction using extended BPEL language.

The initial model of interacting web services was extended using standard XML elements. A recursive method was detailed to calculate service times and response times of processes. However, in order to estimate throughput or resource utilization, a workload model is required. An appropriate workload model can define the arrival rate, service rate and waiting time of a process. Future work will include the introduction of a workload model.

References

[1] Business Process Execution Language for Web Services, ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf

[2] Modeling Extended BPEL Language, Csaba Legány, Márk Kaszó

[3] Rai Jain, The Art of computer performance analysis, Whiley, 1991

[4] Web Services Description Language (WSDL) 1.1, http://www.w3.org/TR/wsdl , March 2001

[5] Extensible Markup Language (XML). http://www.w3c.org/XML