MISQ: A UML-based Analytical Modeling Methodology for Optimizing Web Service Composition

Seog-Chan Oh
IE / Penn State University
sxo160@psu.edu

Dongwon Lee
IST / Penn State University
dongwon@psu.edu

Soundar Kumara
IE / Penn State University
skumara@psu.edu

Abstract

A novel UML-based analytical modeling methodology, named MISQ, is presented for optimizing web service composition in Business Service Networks. MISQ enables functional and temporal analyses at a high-level design stage so that web service composition can be systematically optimized. Furthermore, MISQ provides an automatic generation of web service implementations for improving productivity and reliability.

1. Introduction

In Business Service Networks (*BSN's*), by combining multiple, heterogeneous "services," one can establish new value-added business processes for further applications. In particular, *web services* have emerged as a popular means to describe the "services" that each vendor provides. In such a setting, one of the key issues is how to generate, discover, compose, and optimize web services that are of interest.

In this paper, we especially focus on the problem of optimizing web service composition and propose a novel methodology, MISQ, as a solution. That is, we use UML to design agent based business processes, and two formal modeling schemes, Stochastic Process Algebra (SPA) and Generalized Stochastic Petri Nets (GSPN) [5], to analyze initial business processes design and to obtain optimized parameters. Finally, we propose to use the Business Process Execution Language for Web Service (BPEL4WS) [7] as implementation artifacts for expressing the optimized business processes.

Example. 1 (Motivation)

Consider a scenario in a *BSN's* where the optimization of composed web services is a crucial issue.

Suppose Bill opens an Internet-based auto loan brokerage company, *FirstBroker*, where he locates a loan with low interest rate for customers who pays a nominal fee as a return. *FirstBroker* uses web services from three loan companies, *StarLoan*, *UnitedLoan*, and *BestLoan*. Once *FirstBroker* gets customer's inquiry, it sends bid requests to three loan companies using their web services, and forwards the lowest interest rate among returned to the customer. Whenever *FirtstBroker* sends loan rate requests to the loan companies, *FirstBroker* has to pay a fee to each. That is, *FirstBroker* is a business adapter and three loan web services are software vendors in the *BSN's*.

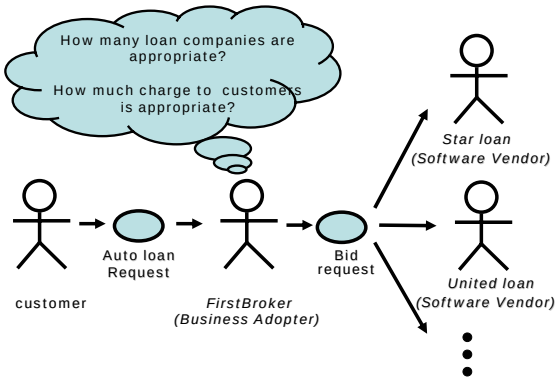


Figure 1. Use case of *FirstBroker* example

Furthermore, a customer pays a fee to *FirstBroker* only if she is satisfied with the proposed rate and decides to make a contract with *FirstBroker*. In summary, Bill's profit model is the following:

Profit model = (# *of accepted proposals by customers* × *charge per customer*) – (# *of loan rate requests* × # *of loan companies* × *charge per loan rate request*)

Suppose Bill agrees to pay $1 for each loan rate request to loan companies, while charging $10 to customers who eventually accept the proposed rate. The business is initially booming, attracting large number of customers due to the fact that customers do

not have to pay for initial inquiries, and pay $10 only afterwards. However *FirstBroker* eventually files a bankruptcy despite many customers submitting inquiries.

The scenario presented often occurs in combining and composing new services in *BSN's* where a decision for parameters must be made to maximize profits. If Bill has chosen a correct number of web services (i.e., loan companies) and proper service charge to customers, possibly he would have been still in business.

Like the case of *FirstBroker*, early identification of optimal values through formal analyses is particularly desirable since the costs of changing the design at a later stage are much higher [4]. However, identifying optimal ones when multiple web services are complicatedly inter-related is a challenging task, since in real applications, such parameters to consider can be many and non-trivial.

Therefore, there is an imminent need for the methodology that systematically and mechanically helps to *model*, *analyze*, and *optimize* web service compositions. For this solution, we propose MISQ in this paper.

2. Overview of MISQ

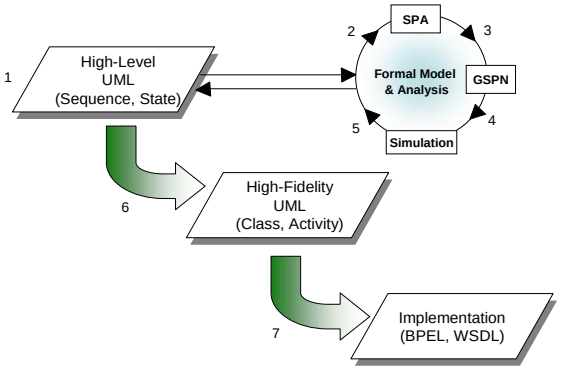


Figure 2. Overview of MISQ

As illustrated in Figure 2, MISQ consists of *analysis* and *implementation* stages. Informally, the analysis stage runs as follows:

1. Design high-level UML diagrams such as state and sequence diagrams.
2. Transform high-level UML designs into a formal model in Stochastic Process Algebra (SPA) model.
3. Transform SPA into Generalized Stochastic Petri-Net (GSPN) model using steps in [5].
4. Perform analysis via simulation.
5. Based on simulation results, identify optimal parameters and design. If needed, 2~4 steps may be repeated.

The implementation stage is adopted from [2] similar to waterfall model of software development. It runs as follows:

6. Based on the optimized high-level design, produce high-fidelity such as class and activity diagrams.
7. From high-fidelity model, generate implementation artifacts.

MISQ contributes the following:

- A Petri-Net model for analyzing initial high level UML based designs, and the temporal and functional analysis for optimization can increase productivity and reliability of web service based software systems in *BSN's*
- A methodology for seamless integration of several languages or modeling tools (e.g., UML, SPA, GSPN, WSDL and BPEL): and a detailed example with a simulation result to illustrate the effectiveness of the proposed methodology.

3. Related work

Our research integrates three different streams of work: deriving analysis model from UML, deriving implementation artifacts from UML and transforming models from SPA to GSPN.

To remedy lacks of verification and validation inherent in UML, some researches [3, 5] tried to translate UML into process algebra. In [3], focus is on a sequence diagram where objects of sequence diagram are considered as π-calculus processes and messages are represented as actions among these processes.

Despite the inherent semi-formality, UML has a strong descriptive power for *high-level modeling* as well as *high-fidelity modeling* [2]. Among UML diagrams, state diagrams and sequence diagrams are sufficient to represent the high-level model. On the other hand, a high-fidelity model is capable of representing the details of implementation artifacts. Usually, a high-fidelity model can be expressed with class and activity diagrams of UML. In [2], the mapping from high-fidelity model to corresponding implementation artifacts is provided using UML 1.4 profile and BPEL4WS [7] as implementation artifacts.

In [6], comparisons between GSPN and SPA with different perspectives are given. In our proposal, we use both GSPN and SPA as an analysis model to optimize web service composition.

4. MISQ Methodology

MISQ is based on various models (i.e., UML, SPA, GSPN, BPEL, WSDL) and transformation procedures between models. In the interest of space, here we only present brief overview of SPA and GSPN.

Definition 1 (SPA)

Stochastic Process Algebra (SPA) is described by the following grammar, [6]

$$P ::= Stop \mid (a, \lambda).P \mid a.P \mid P+P \mid P \|_s P \mid P \setminus S \mid Q$$

where a variable P, Q, … denotes process variables, while S is a set of synchronization actions. The intuitive meaning of these elements is:

- *Stop* denotes the halting process.
- The process $(a, \lambda).P$ models a *delayed* process that performs the action a with *delayed* rate λ and then behaves as process P.
- The process $a.P$ models an *immediate* process that performs the action a without any delay and then behaves as process P.
- The *choice* operator '+' is used to model alternative behavior.
- The *parallel* operator '$\|_s$' models the parallel execution of two processes which have to synchronize in actions within the set of synchronizing actions S.
- The *hiding* operator ' \ ' is used for declaring actions as internal, and often used to abstract away from internal events.

Definition 2 (GSPN)

Generalized Stochastic Petri-Net (GSPN) [6] is defined as a 5-tuple (PL, T, W, M_0, L), where:

- PL is a finite set of places.
- T is a finite set of transitions partitioned into two sub-sets T_I (immediate) and T_D (delayed) transitions, where, transitions, $t \in T_D$ are associated with *delayed* rate λ.
- $W \in (PL \times T) \cup (T \times PL)$ is a set of directed arcs (i.e., flow relation).
- M_0: $PL \rightarrow \{0, 1, 2, \ldots\}$ is the initial marking.
- L: $T \rightarrow \Lambda$ is a labeling function where Λ is a set of operation names.

Example. 2 (SPA & GSPN)

Consider the example scenario of Example 1 again. A customer checks the proposal of *FirstBroker* and either accepts or reject the proposal. Since the customer chooses one behavior between two choices, we represent this process with *choice* operator of SPA, '+', as follows:

$$choice\_decision := (accept + reject).$$

Similarly, we can map *choice_decision* into GSPN model as shown in Figure 3. Here, the place, *choice_start* with a token enables both *accept* and *reject* transition. If *accept* transition is fired, the token switches places from *choice_start* to *accept_decision*. On the contrary, if *reject* transition is fired, the token switches places from *choice_start* to *reject_decision*.

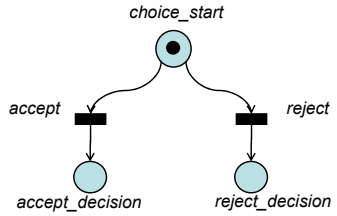


Figure 3. The process *choice_decision*

Definition 3 (MISQ Model)

A MISQ Model is an 8-tuple (*DSequence*, *Agent*, *Protocol*, *DState*, *DClass*, *DActivity*, *SPA*, *GSPN*) where:

- *DSequence* is a sequence diagram with objects, behaviors and messages between objects.
- *Agent* is a set of objects of *DSequence*. We denote each element of *Agent* as $a(i)$ with i being the position of the element (i.e., if $|Agent| = n$, $a(1)$ and $a(n)$ are the leftmost and rightmost objects in *DSequence*).
- *Protocol* is a set of protocols. Individual protocols is a set of messages between $a(i)$ and $a(j) \in A$ where $i < j$ and denoted as $prot(i, j)$.
- *DState* is a set of state diagrams. We denote each element of *DState* as $ds(i)$ which is the state diagram of $a(i) \in A$.
- *DClass* is a set of stereo-typed class diagrams such as *DClass-dependency*, *DClass-datatype*, *DClass-interface*, *DClass-protocol*, *DClass-process*. *DClass-dependency* defines the dependency relationship between each element in *Agent*. *DC-datatype* defines message contents and data classes as well as the relationship between message contents and data classes. *DC-interface* defines operations. *DC-protocol* defines roles of corresponding port type. *DC-process* defines internal variables and its ports connected to each element in *Agent*.
- *DActivity* defines activity diagrams for element in *Agent*.

Next, we present several transformation procedures from one model to another in MISQ. Due to the limited spaces, correctness proofs of the procedure are omitted. Let us first lay down a few assumptions.

1. $a(i) \in A$ $(i > 1)$ has communication with the left and right objects, that is, $prot(i\text{-}1, i) \neq \varnothing$ and $prot(i, i\text{+}1) \neq \varnothing$. For example, $a(1)$ has $prot(1, 2) \neq \varnothing$, and $a(n)$ has $prot(n\text{-}1, n) \neq \varnothing$.
2. For $prot(i, j)$, $|i - j| \leq 1$. That is, each object communicates only with its immediate neighbors.
3. $|A| \geq 2$. That is, there are at least two objects.

Now, we present three transformation procedures: (1) UML to SPA, (2) SPA to GSPN, and (3) UML to Implementation.

Procedure 1 (UML to SPA)

In this procedure, the given UML is re-captured into SPA model. It has two steps.

1. *Building Atomic processes*
   - 1.1 Prepare *DSequence*, *Agent*, *Protocol*, *DState*.
   - 1.2 Create $APset = \{x \mid x \in SPA\} = \varnothing$.
   - 1.3 Set $i = 1$ and choose an $a(i) \in A$.
   - 1.4 Create an atomic process, $p(i) \in SPA$.
   - 1.5 Start transforming $ds(i)$ into $p(i)$. Transitions of $ds(i)$ are transformed to either *delayed* or *immediate* actions. If a transition does not have any temporal information, it becomes immediate action. 'a'. Otherwise, λ is added and becomes the *delayed* action (a, λ).
   - 1.6 If any action branch exists, it is expressed by a non-deterministic choice; '+'.
   - 1.7 A sequence of transitions in $ds(i)$ corresponds to the sequence of actions in $p(i)$.
   - 1.8 $APset = \{p(i)\} \cup APset$.
   - 1.9 If $|APset| = |Agent|$, that is, all $ds(i) \in DState$ is transformed, then the procedure stops. Otherwise, increase i by 1 and go to step 1.3.
2. *Building a Composite process*
   - 2.1 Create a process, $System \in SPA$ and $System := p(1)$. Increase i to 2.
   - 2.2 Choose $p(i) \in APset$.
   - 2.3 $System = System \|_S P(i) \setminus S$, where $S \equiv prot(i\text{-}1, i)$.
   - 2.4 If $i = |APset|$, that is, all the $p(i)$ get combined into *System*, then stop. Otherwise, increase i by 1 and go to step 2.2.

Procedure 2 (SPA to GSPN)

In this procedure, the SPA model is transformed into Petri-Net based GSPN graphical model for easier manipulation. As shown in Figure 4, It is generally known that any SPA model can be represented as a GSPN model and details of such translations can be found in [3, 6]. In our proposal, the approach introduced in [5] is used, for instance. Due to lack of space, the entire procedure cannot be described.

Procedure 3 (UML to Implementation)

Once the high-level UML design has been optimized in the GSPN model, finally, web service implementation can be generated in this procedure. We use the methods in [2], but can use other implementation-specific method for this procedure (e.g., from UML to CORBA)

1. Based on optimized system specification obtained in Procedure 2, *DClass-dependency*, *DClass-datatype*, *DClass-interface*, *DClass-protocol*, *DClass-process*, and *DActivity* are drawn.
2. *DClass-dependency* maps to an XML namespace import in WSDL. *DClass-datatype* maps to message types and data types in WSDL. *DClass-interface* maps to operations types in WSDL. *DClass-protocol* maps to port and service link types in WSDL. *DClass-process* and *DActivity* map to process definitions in BPEL.

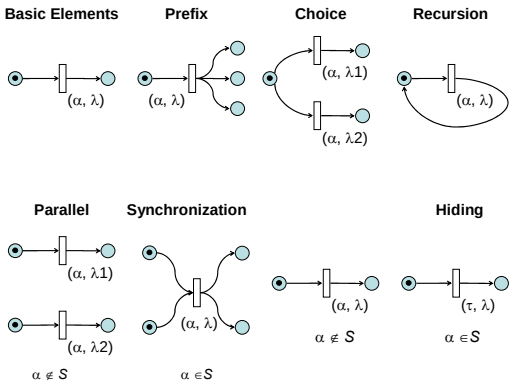


Figure 4. Mapping SPA operations into GSPN

5. Illustrative example

In this section, let us demonstrate how to optimize web service composition using the MISQ methodology. Table 1 summarizes notations used in this example.

Table 1. Notations for the example

Notation	Meaning
C	Customer, C's inter-arrival time follows exp(λ).
B	Brokerage web service.
WS	A set of auto loan web services, $ws_j \in WS$, $1 \leq j \leq n$. We assume that $1 \leq n \leq 4$.
Rate(*ws*)	A loan rate returned from $ws \in WS$, uniform(5, 6) is followed.

t_o	Time-out until which B waits for $Rate(ws)$
$WS(S)$	A set of web services, $WS(S) \subset WS$, that successfully send a loan rate before t_o
$WS(F)$	A set of web services, $WS(F) \subset WS$, that fail to send a loan rate before t_o
$Min(Rate)$	Smallest $Rate(ws_j)$, $\forall\ ws_j \in WS(S)$
$Fee(ws_j)$	Service fee that B pays to $ws_j \in WS(S)$
$Fee(B)$	Service fee that C pays to B
AR	*Accept rate*[1], $AR = exp\{-\sigma \times (Min(Rate) - 5)\} - 2^{(Fee(B)-10)}/2^{10}$, where σ is a preference parameter
PT	Profitable throughput $PT = \|C\| \times AR$

PT is exponentially decreased as $Min(Rate)$ increases (i.e., customers will not accept the offer if the rate is high), and also decreased in proportion of $2^{(Fee(B)-10)}/2^{10}$ as $Fee(B)$ increases (i.e., customer will not accept the offer if the service charge to B is high).

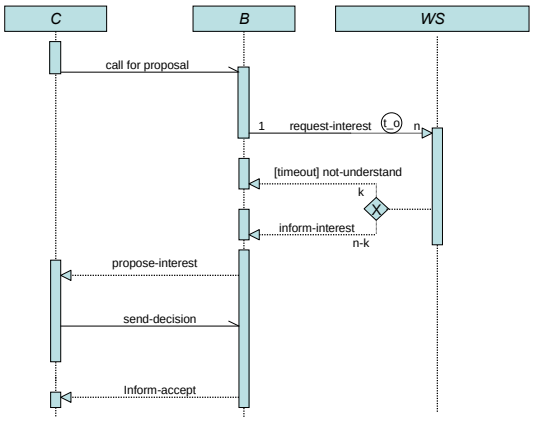


Figure 5. Sequence diagram of the example

5.1. Scenario

Consider the following scenario:

1. C seeks for an auto loan with minimum interest rate, sends an inquiry to B (C has no direct access to WS).
2. B relays the C's request to each $ws_j \in WS$.
3. ws calculates and returns its $Rate(ws_j)$ to B.
4. The communication between B and ws_j is asynchronous with the time-out, t_o. After t_o, B does not wait for $Rate(ws_j\ s)$ anymore. B must pay $Fee(ws_j)$ to successful ws_j who returns $Rate(ws_j)$ within t_o.
5. B sends $Min(Rate)$ to C.
6. If C accepts $Min(Rate)$, C pays $Fee(B)$ to B. Otherwise B cannot charge $Fee(B)$ on C.

[1] *AR* expresses C's purchasing intention whose parameters could be selected based on real market surveys. Here, however, we simply use parameters, *exp* and 2, in the interest of time.

Figure 5 illustrates the sequence diagram of the scenario.

5.2. Applying MISQ to the example

We want to "maximize" the expected profit of B who is a business adopter in the context of *BSN's*. Thus, the objective function, Z, representing the expected profit of B can be: $Z = Fee(B) \times (PT) - Fee(ws) \times |WS(S)| \times T$. If $Z \geq 0$, B makes a profit. Z is directly proportional to PT. If $|WS|$ increases, PT is likely to increase because C has a better chance to obtain lower $Min(Rate)$ but B has to pay more fees to increased $|WS(S)|$. Meanwhile, if $Fee(B)$ decreases, PT may increase since low service charge can attract more C to accept the offer but B's profit decrease.

Note that there are two trade-off relations we need to find the optimal values as followings:

- $|WS| = n$: How many web services of loan companies are economical for B to use?
- $Fee(B)$: How much service charge on customers are appropriate?

Since we assumed $1 \leq n \leq 4$, we apply MISQ analysis starting with $n = 1$ and can repeat the analyses by increasing n by 1. If $n = 1$, $Agent = \{a(1), a(2), a(3)\}$ where $a(1)$ is C, $a(2)$ is B and $a(3)$ is each ws_j of WS. Similarly, $DSequence = \{ds(1), ds(2), ds(3)\}$ where $ds(1)$, $ds(2)$ and $ds(3)$, and is shown in Figure 6. $Protocol = \{prot(1,2), prot(2,3)\}$ where $prot(1,2)$ ={call for proposal, propose-interest, send-decision, inform-accept} and $prot(2,3)$ = {request-interest, not-understand, inform-interest}.

5.2.1. Building atomic and composite processes. We can first build the following atomic processes:

- *customer*:= call-for-proposal; propose-interest; (accept + reject); send-decision; inform-accept; *throughput*.
- *broker*:= call-for-proposal; request-interest; (not-understand + inform-interest); propose-interest; send-decision; inform-accept; *broker*.
- *loan*:= request-interest; ((timeout, t_o);not-understood + (service-done, μ);inform-interest); *loan*.

In addition to original atomic processes, we can add two more processes; arrival and buffer, for collecting analysis data as follows:

- *arrival* := (gen, λ); call-for-proposal; *arrival*.
- *buffer*(i):= (gen, λ); *buffer*(i+1) + inform-accept; *buffer*(i-1), where $i \geq 1$

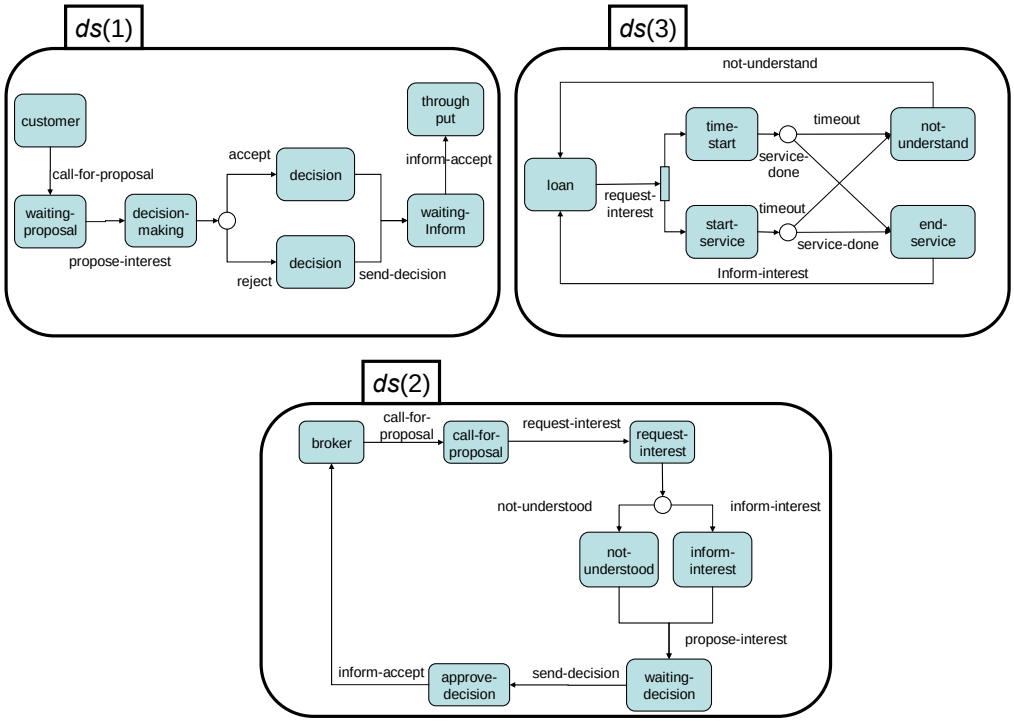


Figure 6. State diagrams of the example

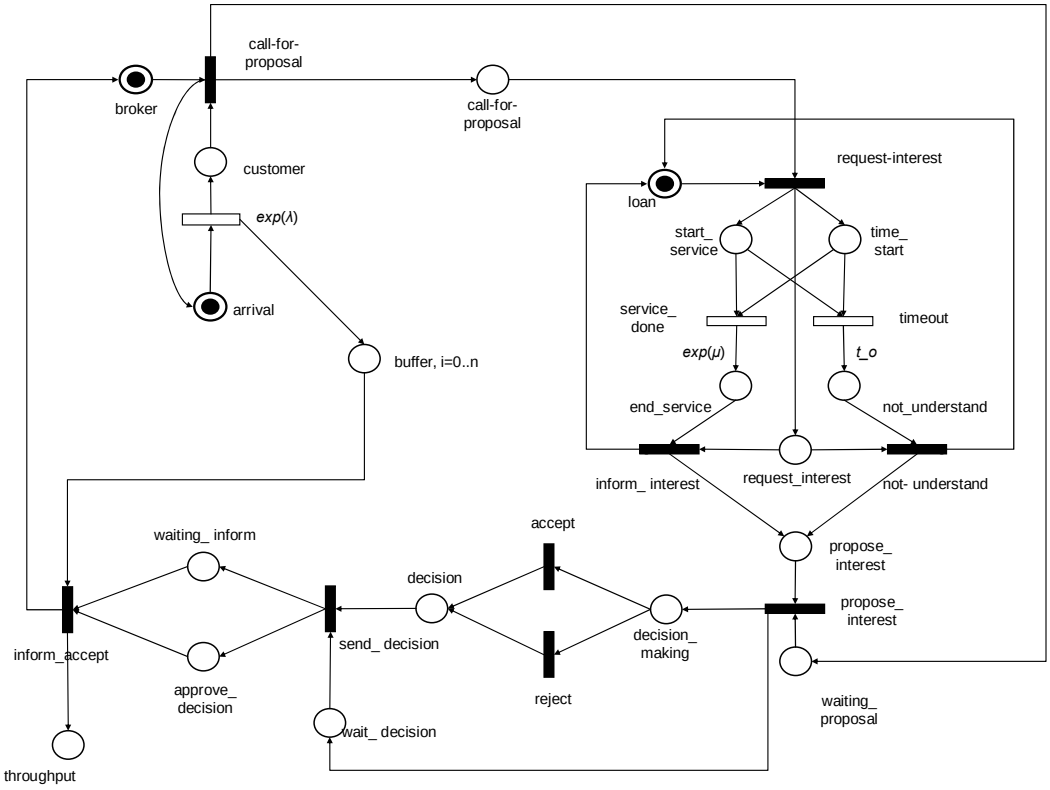


Figure 7. GSPN of *System*

Next, based on the aforementioned atomic processes, we build the composite process of *System* as follows:

- *System* := (*System'* ||S_1 *arrival*) ||S_2 *buffer* \ ($S_1 \cup S_2$) where S_1 = {call-for-proposal}, S_2 = {gen , propose-interest}

5.2.2. Transforming SPA into GSPN. Through the SPA to GSPN procedure, the composite process System, in SPA is transformed into GSPN as shown in Figure 7.

5.2.3. Simulation of GSPN

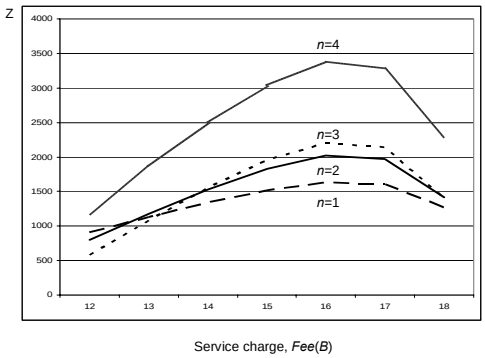
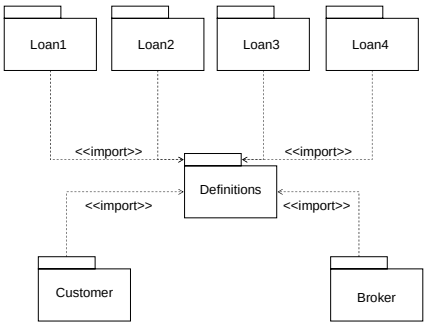


Figure 8. Profit according to |*WS*| and *Fee*(*B*)

We conducted simulations for four experimental cases; |*WS*| = 1, 2, 3, and 4. We assumed that $1/\lambda = 1/\mu$ = t_o = 4 hours, σ = 5, and Fee(wsj) = $1. GSPN model simulation was done using HPSim [1] and the result analysis was conducted with MS Visual Basic and Excel. Simulation time was set to same as *B*'s life cycle, 10,000 hours.

As shown in Figure 8, The optimal setting for the scenario occurs when |*WS*| = 4, *Fee*(*B*) = $16 with the expected profit of *B* being $3,373.

5.2.4 High fidelity UML and implementation



- *System'* := *customer* $\|_{prot(1,2)}$ *broker*$\|_{prot(2,3)}$ *loan* \ (*prot*(1,2) ∪ *prot*(2,3))

Figure 9. Dependency Diagram of the example

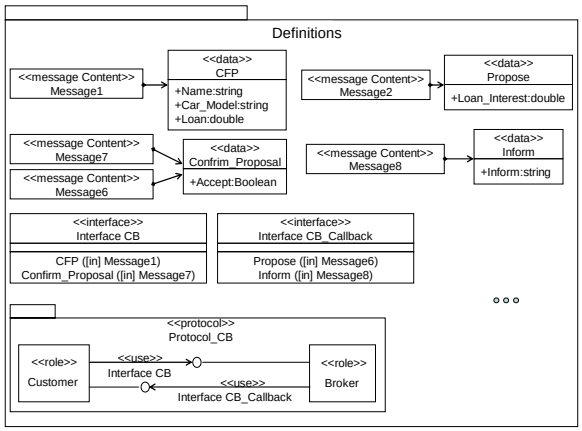


Figure 10. Definitions package of the example

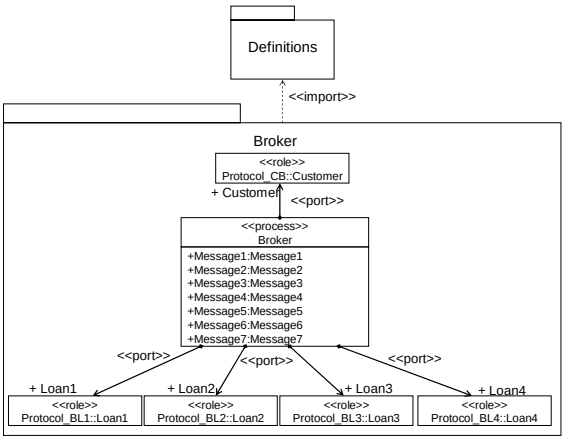


Figure 11. Broker package of the example

Once we acquire optimal parameters for the auto-loan example, we can build *DClass-dependency* as in Figure 9. Similarly, we also can generate *DClass-datatype*, *DClass-interface* and *DClass-protocol* as in Figure 10. Those models maps into a WSDL file. Furthermore, we can also build, *DClass-process* in Figure 11, and *DActivity* (omitted due to space limitation), and those models map into a BPEL file. Due to lack of space, the entire diagrams and codes of WSDL cannot be presented. Instead, some part of implementation codes of WSDL and BPEL are illustrated in Figure 12 and 13 respectively.

```
<?xml version="1.0"?>
<definitions name="Broker" ... >
```

```
<types>
 <element name = “CFP">
                <seqeunce>
                                <element name=“Name” type=“string”>
                                <element name=“Car_Model” type=“string”>
                                <element name=“Loan” type=“int”>
                </sequence>
 <element>
...
</types>
<message name=“Message1">
                <part name="parameters“ element="CFP"/>
</message>
...
<portType name=“Interface CB">
                <operation name=“CFP">
                                <input message=“Message1"/>
                </operation>
                <operation name=“Confirm_Proposal">
                                <input message=“Message7"/>
                </operation>
</portType>
...
<serviceLinkType name=“Protocol_CB">
                <role name=“Customer">
                                <portType name=“Interface CB_Callback"/>
                </role>
                <role name=“Loan">
                                <portType name=“Interface CB”/>
                </role>
</serviceLinkType>
...
</definitions>
```

Figure 12. WSDL of the example

```
<process name =“Broker“ … >
<partners name=“Customer” serviceLinkType=“Protocol_CB”
                partnerRole=“Protocol_CB:Customer”
                myRole=“Protocoal_CB:Broker”/> </partners>
...
<receive name=“customerInput” partnerLink=“Customer"
portType=“Interface CB"    operation=“CFP" variable=“Message1" ... />
<flow>
<sequence>
<invoke name=“Loan1Invoke” partnerLink=“Loan1”    portType=“Interface
BL1"
operation=“CFP" variable=“Message1" ... />
<recevie name=“Loan1Invoke” partnerLink=“Loan1”    portType=“InterfaceBL1
Callback" operation=“Propose" variable=“Message3" ... />
</sequence>
...</flow>
<assign name=“InterestAssign” >
<copy>
<from variable=“message2” portion=“LoanInterest” />
<to variable=“message6” portion=“LoanInterest” ><copy/>
</assign>
<switch>
<case condition=“message6/LoanInterest > message3/LoanInterest”>
<assign name=“Loan2Assign” >
<copy><from variable=“message6” portion=“LoanInterest” />
<to variable=“message3” portion=“LoanInterest” ><copy/>
</assign>
<otherwise><empty /></switch>
...
<invoke name=“Propose” partnerLink=“Customer” portType=“Interface CB
callback "
operation=“Propose " variable=“Message6" ... />
<receive name=“ConfirmProposal” partnerLink=“Customer"
portType=“Interface CB“ operation=“ConfirmProposal "
variable=“Message7" ... />
<invoke name=“Inform” partnerLink=“Customer”
portType=“Interface CB callback“ operation=“Inform " variable=“Message8" ...
/>
</process>
```

Figure 13. BPEL of the Broker

Figure 13 illustrates the BPEL of the example which imports the WSDL and orchestrates web services including customer and four loan web services. The main body of the BPEL is <process> which can be divided into two parts such as the process type definition and the process activity definition.

6. Conclusion

The MISQ systematically optimizes web service composition to identify the optimal values such as the number of ideal web services, maximum throughput, etc. There are several future research directions. In addition to simple value optimization, more functional analysis (e.g., deadlock detection or security flaw detection) can be greatly benefited by MISQ. Also, considering real-time IT provisioning and adoption enabled by *BSN's*, more "dynamic" optimization is a challenging goal. For instance, optimizing the dynamic workflow [8] of web service components can greatly benefit both software vendors and business adopters. Toward this scenario, discovering, dynamically composing, and optimizing large-scale (e.g., in the range of 1,000 - 10,000) web services is a challenging problem. In EEE05 [9], we approached the problem by viewing web services composition as a graph search problem. What has presented in this paper is thus complementary to [9].

In the near future, we plan to combine the ideas of [9] and that of MISQ to accomplish truly dynamic web service composition methodology.

7. References

[1] H. Anschuetz, "HPSim Copyright © 1999-2001", Available: http://www.winpesim.de/petrinet/e/hpsim_e.htm.

[2] J. Amsden, T. Gardner, C. Griffin, and S. Iyengar, "Draft UML 1.4 Profile for Automated Business Processes with a mapping to BPEL 1.0", *IBM*, June 2003.

[3] K. Korenblat, and C. Priami "Extraction of π-calculus specifications from UML sequence and state diagrams", *Technical Report DIT-03-007*, *Informatica e Telecomunicazioni, University of Trento*. 2003.

[4] M. Marzolla, "Simulation-Based Performance Evaluation of Software Archiecture described in UML", *Universita Ca Foscari di Venezia*, November 2002.

[5] M. Ribaudo, "Stochastic Petri Net Semantics for Stochastic Process Algebra" *Proceeding of the 6th Int'l Workshop on Petri Nets and Performance Models*, 1995.

[6] S. Donatelli, H. Hermanns, J. Hillston, and M. Ribaudo, "GSPN and SPA compared in Practice", *Quantitative Modelling in Paralle Systems*, Springer, 1995.

[7] T. Andrews et al, "Business Process Execution Language for Web Services (BPEL) 1.1", *OASIS*, May 2003.

[8] J. Kim et al, "Web Services and BPEL4WS for Dynamic eBusiness Negotiation Processes", *Proceeding of the Int'l Conf. on Web Services*, 2003.

[9] S. Oh, B. On, E. J. Larson, and D. Lee, "BF*: Web Services Discovery and Composition as Graph Search Problem", *Proceeding of IEEE Int'l Conf. on e-Technology, e-Commerce and e-Service (EEE)*, Hong Kong, China, March 2005.