Research Journal of Applied Sciences, Engineering and Technology 4(22): 4905-4913, 2012
ISSN: 2040-7467

Submitted: June 09, 2012 Accepted: July 09, 2012 Published: November 15, 2012

A Formal Modeling and Implementation of Particle Swarm Optimizer for QoS-Aware Service Selection with an Extended Pi Calculus

[1]Desheng Li and [2]Na Deng
[1]School of Science, Anhui Science and Technology University, Fengyang 233100, China
[2]School of Computer, Hubei University of Technology, Wuhan 430068, China

Abstract: For past years, Particle Swarm Optimization (PSO), one of the evolutionary computational techniques, has been intensively studied and applied in both academia and industry. Recently there has been a shift from consideration of design of concrete algorithms to a consideration of the formalization models of optimization approaches. However, the meta-search procedure is not the primitive of the algebra, which not participates in the derivation of the inference of expressions. For this reason, the models above could not be seen as a unity in a strict mathematics form. Moreover, the operators of traditional algebra limit the express of complicated processes, such as concurrent patterns. As a result, the cost calculation of the whole process is not an easy thing only according to the algebraic form itself. Attempting to solve these issues, a new formal modeling of particle swarm optimizer from a perspective of an extend version of Pi calculus is proposed in this study, which treats the whole operations in PSO as a kind of meta-search procedure and owns the cost operator and other operators supporting concurrent patterns. On the basis of this algebra, the QoS-aware service selection problem can be seen as a particular cost derivation under the LTS semantics. Based on the theoretical model, a simulator with a core of Pi calculus compiler is developed to verify our theory and also show the practical applicability in a real scenario.

Keywords: Particle swarm optimization, pi calculus QoS, service selection

INTRODUCTION

Evolutionary algorithm is used widely for solutions of polynomial and intractable optimization problems and it can be understood as a multi-agent competitive probabilistic search essentially. Particle Swarm Optimization (PSO) is one of the evolutionary computational techniques proposed in literature (Kennedy and Eberart, 1995), which models the social behaviors, such as movement and collaboration of birds group for food. For past years, PSO has intensively studied and applied in both academia and industry. For past years, PSO has been intensively studied and applied in both academia and industry.

Recently there also has been a shift from consideration of design of concrete search algorithms to a consideration of the formalization models of general approaches. For the modeling of the system and behaviors of intelligent system, a lot of researches can be found as follow: Weighted Synchronous Calculus of Communicating Systems (WSCCS) (Tofts, 1991), a process algebra, was used by Tofts to model social insects and its application of conjunction with a dynamical systems approach for analyzing the non-linear aspects of social insects in Sumpter *et al*. (2001); Dynamic Emergent System Modeling Language (DESML), a variant of UML (Kiniry, 1998), has been proposed for modeling emergent systems. In Eberbach (2003), several super Turing models have been discussed, including cellular automata, Interaction Machines, Persistent Turing Machines and process calculi, such as the Pi-calculus and $-calculus. Although lots of modeling approaches are proposed for intelligent algorithms, but there still exist several pending problem to be solved:

- The meta-search procedure is not the primitive of the algebra, which not participate in the derivation of the inference of expressions. For this reason, the models above could not be seen as a unity in a strict mathematics form.
- Then operators of traditional algebra limit the express of complicated processes, such as concurrent patterns. As a result, the cost calculation of the whole process is not an easy thing only by the algebraic form itself.

Attempting to solve these issues, we present a new process algebra Pi-beam-cost, along the rout of the $-calculus and Pi-calculus, to discuss the modeling of PSO and its scheme for QoS-aware service selection. The goal of the Pi-beam-cost calculus is to propose a

Corresponding Author: Desheng Li, School of Science, Anhui Science and Technology University, Fengyang 233100, China

computational model with built-in concurrency with constraints structures, logic summations operations, QoS cost measure and the evolutionary search operations. On the basis of this algebra, the QoS-aware service composition can be seen as a particular cost derivation under the LTS semantics. Finally, we will introduce our design and implementation of the simulator with the tests on a real scenario.

THE PI-BEAM-COST CALCULUS

Language: Let $a, b, \ldots, \vec{x}, \vec{y}, \ldots, \vec{x}_m^n, \vec{y}_p^q, \ldots$ be names, while $P, Q, R, \ldots$ ranged over the expression set $\mathcal{P}$. In Pi-beam-cost calculus, everything is a cost expression: agents, environment, communication, interaction links, inference engines, modified structures, data and code. Pi-cost-expressions can be divided into 2 kinds, i.e., simple expressions which are considered to be atomic in evolution and composite expressions which consist of distinguished components and can be interrupted. Moreover, expressions return other cost expressions as the result of evaluation.

Definition 1 (Pi-beam-cost calculus): The set $\mathcal{P}$ of Pi-beam-cost calculus process expressions consists of simple Pi-cost-expressions α and composite expressions $P, Q, R, \ldots$, which can be defined by the following syntax:

$$\alpha ::= \vec{x}_m^n(\vec{y}_n).P \mid \overline{\vec{x}_m^n} < \vec{y}_n > .P \mid \perp \mid \varepsilon \mid \$(P) \mid \tau.P \mid (\nu\, x)P \mid !_k.P$$
$$P ::= P \mid Q \mid P \overset{*}{\|} Q \mid P \overset{+}{\|} Q \mid P \overset{\oplus}{\|} Q \mid P \overset{\Delta}{\|} Q \mid P \overset{\nabla}{\|} Q \mid P * Q$$
$$\mid P + Q \mid P \oplus Q \mid P \Delta Q \mid P \nabla Q \mid A(\vec{X}_n)$$

We write empty parallel composition, general, cost and adversary choices as $\perp$ denoting blocking and empty sequential composition as ε, which is used to mask, make invisible part of Pi-cost - expression. It is a process which blocks and does nothing (degenerate process or inaction with blocking). $\perp$ Also can be modeled as the logical false, any Pi-cost–expression different with $\perp$ represents true. Unlike $\perp$, the ε never blocks, but executes silently and also returns nothing, which may often be considered to be unobservable. But note that the effects of silent actions can be estimated by costs. It can be used to shut down a part of the expression and make it invisible for a given agent. $!_k P$ (Replication) denotes a number of copies of in parallel. The cost function $\$P$ calculates a cost associated with process P, then return a value typically defined in domain of real numbers $\mathcal{R}$ with added infinity ∞. A standard cost function is predefined in Pi-cost–calculus, which can be changed to user-defined ones with few modifications, just as our QoS cost functions in the latter section. $\vec{x}_m^n(\vec{y}_n).P$ denotes the process ready to transmit an input message $\vec{y}_n$ along the channel $\vec{x}_m^n$. $\overline{\vec{x}_m^n} < \vec{y}_n > .P$ is the process ready to transmit $\vec{y}_n : \vec{T}_n$ along $\vec{x}_m^n$. $\tau.P$ denote that after the execution of internal action τ the behavior still appear as P. $(\nu\, x)P$ restricts the name x in P. $P \mid Q$, $P \overset{*}{\|} Q$, $P \overset{+}{\|} Q$, $P \overset{\oplus}{\|} Q$ can be seen a family of the parallel compositions of P and Q. However, P|Q denotes the common parallel of P and Q, usually used to link the components of system simply; and the latter 3 notations not only denote the processes are in parallel, but also assign their logic relations about convergence of them. A cost choice PΔQ selects exactly one with minimal costs and the adversary choice $P \nabla Q$ selects exactly 1 with maximal costs for evaluation. The general choice $P \Delta Q$ selects randomly or by condition exactly one process P or Q.

The indexing set I is a countable infinite set. If I is empty, we write empty three kinds of parallel compositions, general and cost choices as $\perp$ denoting blocking and empty sequential composition as ε, which means invisible transparent action making invisible parts of Pi-beam-cost calculus.

Meta-control procedure of PSO: Let's define some auxiliary notions used in the PSO-optimization meta-search. $Swarm_i[t]$ denote the solution expression for the i-th agent(the swarm) in t-th iteration; While $PSO_i[t]$ is its own search procedure. Herein, we just consider the case of one swarm, so the argument i is often ignored. P_j^t is the current position of the i-th particle in t-th iteration. V_j^t is the current velocity of the i-th particle in t-th iteration; $Plbest_j^t$ is the local best position of i-th particle in t-th iteration; $Pgbest_j^t$ is the global best position of the local best positions of all i-th particle.

Definition 2 (PSO-optimization meta-search procedure): The PSO-optimization meta-search procedure $PSO[t]$ from a swarm of particle population and working in the time generations $t = 0,1,2,\ldots$ is a meta-search procedure Pi-cost-expression consisting of some operation Pi-cost-expressions such as $init[t]$, $eval[t]$, $localbest[t]$, $mut[t]$, $globalbest[t]$, $upd[t]$, $loop[t]$ 和 $exec[t]$ and constructing solutions and the input of them, $Swarm[t]$. The swarm acts with the following procedure $PSO[t]$ in the iterations $t = 0,1,2,\ldots$ where loop meta-Pi-cost-expression takes the form of termcond - eval - localbest - globalbest-upd cycle performing the PSO-Optimization until the goal is satisfied.

The main procedure of PSO can be defined as follows:

$$(PSO[t], Swarm[t]) = ((init(PSO[0], Swarm[0]) \,||\, (loop(Swarm[t+1])))$$

$$loop(Swarm[t]) = ((((\overline{goal}(PSO[t], Swarm[t])) \,||\, (sel(Swarm[t])) \,||\, (\$(Swarm[t])) \,||\, (localbest(Swarm[t])) \,||\, (mut(Swarm[t])) \,||\, (globalbest(Swarm[t])) \,||\, (upd(Swarm[t])) \,||\, (loop(Swarm[t+1]))) \oplus (goal(PSO[t], Swarm[t])) \quad))$$

Firstly, $Swarm[t] =|_{j \in J} P_j^t$ is defined to denote a particle with nearest distance to global best position in P_j^t s.

- The default object is to find a cost expression pair $(PSO[t], Swarm[t])$ to make the objective expression: $goal(PSO[t], Swarm[t]) = \$(PSO[t], Swarm[t]) = \$(Swarm[t])$ satisfy a certain boolean expression: $Termin()$. Hither, the owning search cost of PSO meta-search is not considered, so the object is only $\$(Swarm[t])$. Hence, the main function in this stage is to perform the boolean operation on $\$(Swarm[t])$ to test the termination condition, rather than evaluating on cost.
- In the choice stage, a known expression is used to substitute the invisible part ε.
- In the evaluation stage, $\$(Swarm[t])$ is used to calculate the cost of expression: $\$(Swarm[t]) = \$(|_{j \in J} P_j^t) = \int_{j \in J} \$P_j^t = Min_{j \in J}(\$(P_j^t))$, where the dynamic evaluation function is Min here.
- In the step of getting local best position, $localbest(Swarm[t])$ calculates each particle's local best position.

$$localbest(Swarm[t]) = localbest(|_{j \in J} P_j^t) =|_{j \in J} localbest(P_j^t)$$

$$localbest(P_j^t) = ch_j^{lbest}(Plbest_j^t).ch_j(P_j^t).LBest(Plbest_j^t, P_j^t) .\overline{ch_j^{Lbest}} < Plbest_j^t >$$

$$LBest(Plbest_j^t, P_j^t) = Plbest_j^t \Delta P_j^t$$

- In the mutual interaction phase, $mut(Swarm[t])$ takes charge of sending the particles' local optima, respectively.
  $mut(Swarm[t]) = mut(|_{j \in J} P_j^t) =|_{j \in J} \overline{ch_{togbest_j}} < Plbest_j^t >$
- In the step of getting global best position, $globalbest(Swarm[t])$ collects all local optima, respectively.

$$globalbest(Swarm[t]) = globalbest(|_{j \in J} P_j^t) = ch_{togbest_1}(Plbest_j^t)... ch_{togbest_n}(Plbest_j^t).\overline{ch_{gbest}} < GBest(Plbest_1^t, ..., Plbest_n^t) >$$

$$GBest(Plbest_1^t, ..., Plbest_n^t) = \Delta_{j \in I} Plbest_j^t$$

- In the update phase, $upd(Swarm[t])$ is executed following the below equation:

$$upd(Swarm[t]) = upd(|_{j \in J} P_j^t) =|_{j \in J} upd(P_j^t)$$

- When the termination condition is satisfied, $goal(PSO[t], Swarm[t])$ expresses that the final solution is obtained.

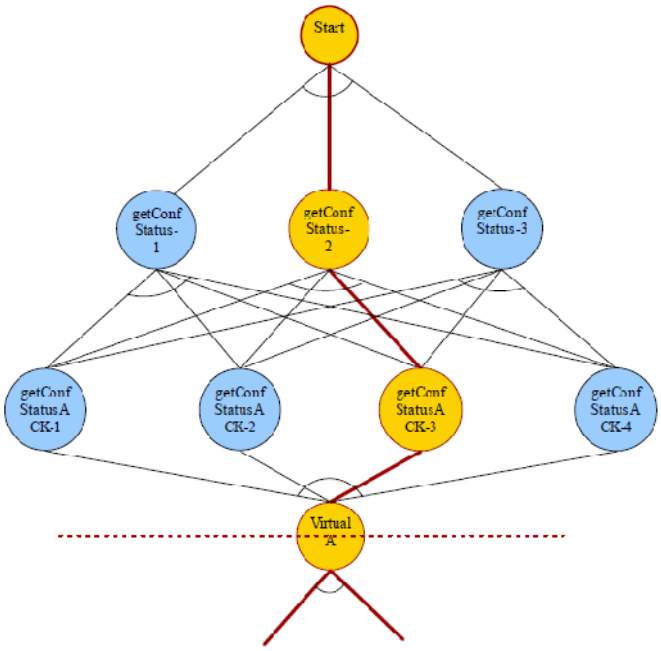

Fig. 1: Fragment of business process flow of "getConfStatus"

THE MODELING OF QOS-AWARE WEB SERVICE SELECTION

Preliminary modeling of qos-aware web service selection with pi-beam-cost: Consider a business process "getConfStatus" in multimedia conference system that supports getting a status of a given conference. The main process has three possible main branches:

- Get the status of a given conference successfully.
- Also get a status when some faults occur.
- Fail to get any message of status when the process time out.
- To explain the principle of utilizing Pi-beam-cost to model the QoS-aware Web service selection problem, we first choose an A*-like algorithm as the preliminary approach, then that based on PSO will be given in the next sub-section. As shown in Fig.1, to make the total executing time minimal, the timeout path has no other physical service to choose. On the other hand, according to equation: $\$(P \oplus Q) = p \cdot \$(P) + (1-p) \cdot \$(Q)$ with fixed p. We can draw a conclusion that the total executing time could reach minimal as long as the delay of Assign B with its corresponding channels gets minimal. Because of the existence of key virtual node Virtual A, the executing time can be divided into to parts: the parts above/below Virtual A. Without loss of generality, we only consider the former to illustrate the cost derivation and generation of LTS.

The system consists of only one agent noted by an expression as follows:

$$P = Start \overset{\Delta}{\|} (GetConfStatus_1 \Delta GetConfStatus_2 \Delta GetConfStatus_3) \overset{\Delta}{|} (GetConfStatus_1 \overset{\Delta}{\|} (GetConfStatusAck_1 \Delta GetConfStatusAck_2 \Delta GetConfStatusAck_3 \Delta GetConfStatusAck_4)) | ((GetConfStatusAck_1 \Delta GetConfStatusAck_2 \Delta GetConfStatusAck_3 \Delta GetConfStatusAck_4) \overset{\Delta}{\|} VirtualA)$$

- Note that, the object is to obtain a LTS tree with minimal cost and its solution. The estimation of expression and ε cost comes from users' experience. Moreover, we assume that the depth of choice in checking phase is 1, while the number of branches is infinitely great, i.e., all branches should be checked. The whole phases of cost derivations are illustrated below and the corresponding final LTS tree is depicted in Fig. 2:

- t = 0, Initialization:
  Initial expression is composed of *Start* and ε_A which denotes an invisible part. *Start* is used to expanded by cost, i.e., $\$(p) = \$(A \| \varepsilon_A) = (0 + 5.0) = 5.0$. LTS tree consists of root state *Start* with cost 0, an empty action ε_A denoting a masked sub tree, including estimated value 5.0 s.
  t = 1, the first iteration.

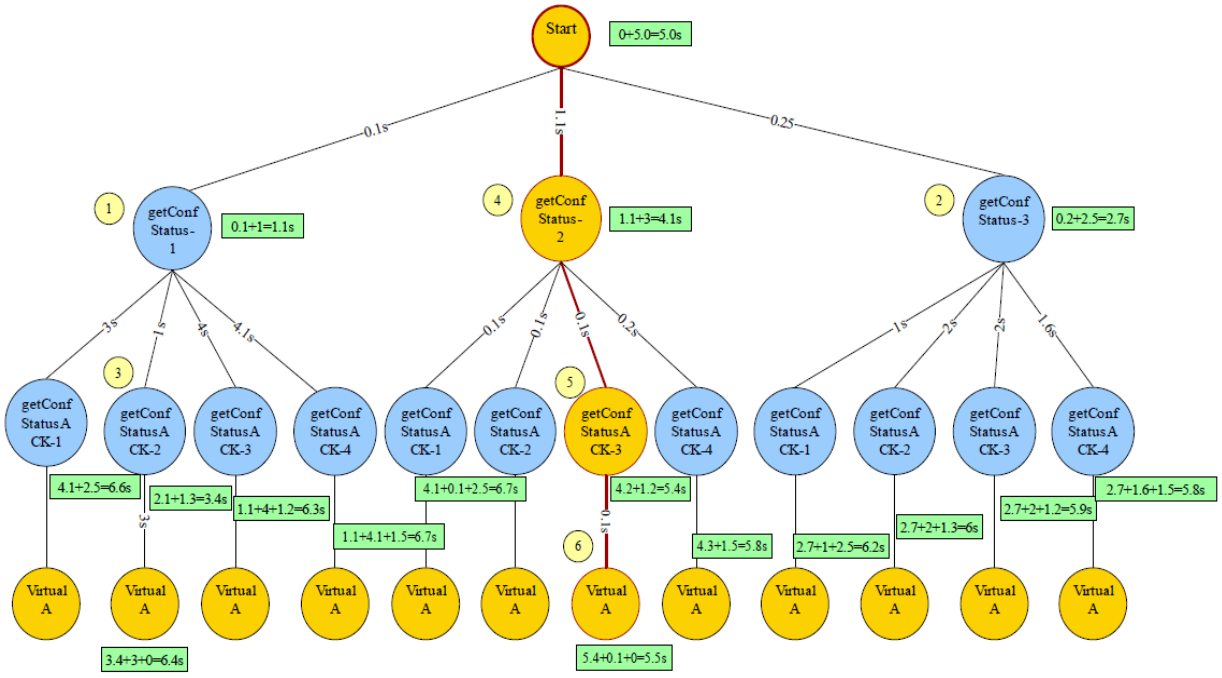

Fig. 2: LTS tree and the cost derivation

Due to *Start* is not the objective state, so a loop phase substitutes ε_A by the use of structural congruence rules. In this phase, the following expression is added onto the LTS tree.

$$\varepsilon_A = (GetConfStatus_1 \parallel \varepsilon_{GetConfStatus_1})\Delta(GetConfStatus_2 \parallel \varepsilon_{GetConfStatus_2})\Delta(GetConfStatus_3 \parallel \varepsilon_{GetConfStatus_3})$$

In other words, we can say it is evolved to:

$$P \Longrightarrow Start \overset{\Delta}{\parallel}((GetConfStatus_1 \parallel \varepsilon_{GetConfStatus_1})\Delta(GetConfStatus_2 \parallel \varepsilon_{GetConfStatus_2})\Delta(GetConfStatus_3 \parallel \varepsilon_{GetConfStatus_3}))$$

Then the total cost value of the LTS tree is calculated in below form. Due to $GetConfStatus_1$ is not the object but a minimum, it is chosen to extended by $\varepsilon_{GetConfStatus_1}$ in children's node in the next generation:

$$\$(P) = 0 + Min(\$(GetConfStatus_1 + 0), \$(GetConfStatus_2 + 0), \$(GetConfStatus_3 + 0)) = Min(0.1 + 1, 1.1 + 3, 0.2 + 2.5) = 1.1$$

- t = 2, the second iteration:
- In this phase, the following expression is added onto the LTS tree.

$$\varepsilon_{GetConfStatus_1} = (GetConfStatusAck_1 \overset{\Delta}{\parallel} \varepsilon_{GetConfStatusAck_1})\Delta(GetConfStatusAck_2 \overset{\Delta}{\parallel} \varepsilon_{GetConfStatusAck_2})\Delta(GetConfStatusAck_3 \overset{\Delta}{\parallel} \varepsilon_{GetConfStatusAck_3})\Delta(GetConfStatusAck_4 \overset{\Delta}{\parallel} \varepsilon_{GetConfStatusAck_4})$$

In other words, we can say it is evolved to:

$$P \Longrightarrow Start \overset{\Delta}{\parallel}((GetConfStatus_1 \overset{\Delta}{\parallel}((GetConfStatusAck_1 \overset{\Delta}{\parallel} \varepsilon_{GetConfStatusAck_1})\Delta(GetConfStatusAck_2 \overset{\Delta}{\parallel} \varepsilon_{GetConfStatusAck_2})\Delta(GetConfStatusAck_3 \overset{\Delta}{\parallel} \varepsilon_{GetConfStatusAck_3})\Delta(GetConfStatusAck_4 \overset{\Delta}{\parallel} \varepsilon_{GetConfStatusAck_4})))\Delta(GetConfStatus_2 \parallel \varepsilon_{GetConfStatus_2})\Delta(GetConfStatus_3 \parallel \varepsilon_{GetConfStatus_3}))$$

Then the total cost value of the LTS tree is calculated in below form. Due to $GetConfStatus_3$ is not the object but a minimum, it is chosen to extended by $\varepsilon_{GetConfStatus_3}$ in children's node in the next generation:

$$\$(P) = 0 + Min(\$(GetConfStatus_1 + Min(\$(GetConfStatusAck_1), \$(GetConfStatusAck_2), \$(GetConfStatusAck_3), \$(GetConfStatusAck_4))), \$(GetConfStatus_2 + 0), \$(GetConfStatus_3 + 0)) = Min(0.1 + 1 + Min(3 + 2.5, 1 + 1.3, 4 + 1.2, 4.1 + 1.5), 1.1 + 3, 0.2 + 2.5) = Min(0.1 + 1 + 2.3, 4.1, 2.7) = 2.7$$

- Also available, after the 5th and 6th iterations, the final node with minimal cost and process expression can be obtained as follows:

$$P \Longrightarrow Start \overset{\Delta}{\parallel}((GetConfStatus_1 \overset{\Delta}{\parallel}((GetConfStatusAck_1 \overset{\Delta}{\parallel} \varepsilon_{GetConfStatusAck_1})\Delta(GetConfStatusAck_2 \overset{\Delta}{\parallel} VirtualA)\Delta(GetConfStatusAck_3 \overset{\Delta}{\parallel} \varepsilon_{GetConfStatusAck_3})\Delta(GetConfStatusAck_4 \overset{\Delta}{\parallel} \varepsilon_{GetConfStatusAck_4})))\Delta(GetConfStatus_2 \parallel ((GetConfStatusAck_1 \overset{\Delta}{\parallel} \varepsilon_{GetConfStatusAck_1})\Delta(GetConfStatusAck_2 \overset{\Delta}{\parallel} \varepsilon_{GetConfStatusAck_2})\Delta(GetConfStatusAck_3 \overset{\Delta}{\parallel} VirtualA)\Delta(GetConfStatusAck_4 \overset{\Delta}{\parallel} \varepsilon_{GetConfStatusAck_4})))\Delta(GetConfStatus_3 \parallel ((GetConfStatusAck_1 \overset{\Delta}{\parallel} \varepsilon_{GetConfStatusAck_1})\Delta(GetConfStatusAck_2 \overset{\Delta}{\parallel} \varepsilon_{GetConfStatusAck_2})\Delta(GetConfStatusAck_3 \overset{\Delta}{\parallel} \varepsilon_{GetConfStatusAck_3})\Delta(GetConfStatusAck_4 \overset{\Delta}{\parallel} \varepsilon_{GetConfStatusAck_4}))) \longrightarrow Start \overset{\Delta}{\parallel} GetConfStatus_2 \overset{\Delta}{\parallel} GetConfStatusAck_3 \overset{\Delta}{\parallel} VirtualA$$

So the result is $\$(P) = 5.5$.

As shown in Fig. 2, the expression unreduced is the generated LTS derivation tree. In Fig. 2, the yellow circle denotes the selected candidate service, while the blue one is discarded. The text in the green box shows the calculation of the total cost from beginning to the current service. After the cease of execution, the final reduced expression is the one of the real executing path tree which is colored as red-line path in Fig. 2 This kind of search is complete in the local finite graph optimization but also with exponential time and space complexity. Thus, in the next sub section, we will give the modeling using meta- heuristic approach such as PSO.

Modeling in PSO scheme: From the above modeling and derivation, we can also describe the QoS-aware service composition by Pi-beam-cost with its built-in PSO-optimization meta-search procedure. Considering different kinds of QoS properties of elements including atomic services and channels between services, a vector composed of QoS properties can be defined as $qos_j = [qos_j^1, qos_j^2, ..., qos_j^n]^T$. Because the criteria have different measures and the different types, so it is necessary to normalized their values into the same interval $[0, 1]$. For a composite service with selected services, the total QoS of it is defined as the aggregated QoS of elements, i.e.,

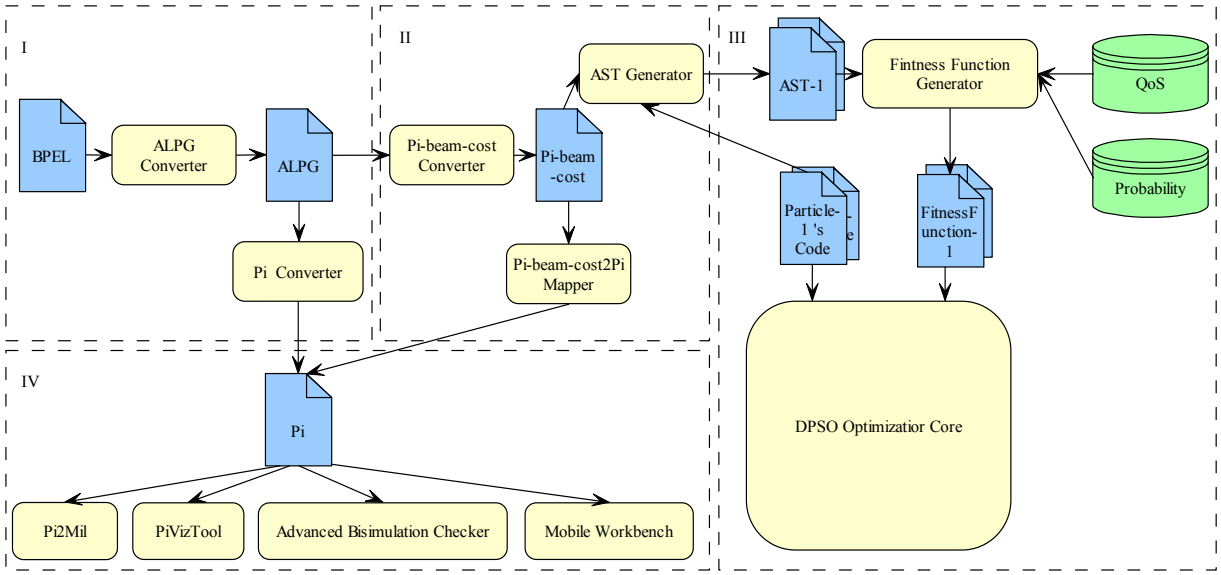

Fig. 3: The architecture of simtool

$$aggqos_j = \Sigma_{j \in J}(weight_j \cdot qos_j)$$

The default goal agent of the QoS-aware web service selection problem is to obtain a pair (PSO[t], Swarm[t]) of Pi-cost, such that:

$$Max\ (\$(PSO[t], Swarm\ [t])) = \$(Swarm[t]) = \$(\cup_{i \in I} P_i^t) = Max_{i \in I}(\$(P_i^t)$$

where, $ is a domain-specific cost function, i.e., the aggregated QoS of the control flow with selected services. Therefore, the problem of QoS-aware web service selection can be reduced into finding a particle with the maximal QoS cost. The evaluation of QoS of a particle based on the algebraic representation will be introduced in the latter part of this section. The related operators from Pi-beam-cost calculus are used to build solutions and the goal is the state with maximal cost.

Therefore, the problem of QoS-aware web service selection can be reduced into finding a particle with the maximal QoS cost. The evaluation of QoS of a particle based on the algebraic representation will be introduced in the latter part of this section. The related operators from Pi-beam-cost calculus are used to build solutions and the goal is the state with maximal cost.

- Initialization: $Swarm[0] = Swarm[0]\Delta\varepsilon_{Swarm[0]}$

The derivation tree starts from its root situation $Swarm[0]$ denoting a swarm of particles and a hidden state sub-tree $\varepsilon_{Swarm[0]}$ representing null continue or future solution may occur. $Swarm[0]$ is the randomly generated n particles in $init(PSO[0], Swarm[0])$. Suppose that $Swarm[0]$ is not the objective state, then $loop(Swarm[t+1])$ is executed to breed next generation of swarm, i.e., $Swarm[1]$ and to discard the $Swarm[0]$. In other words, this procedure can be seen as erasing it by an invisible term $\varepsilon_{Swarm[0]}$.

- t = 1, the first iteration.
  Due to not satisfying the requirement of objective state, then subsequent operations are performed. Firstly, a null substitution is taken to check the swarm's cost, that is $\$(Swarm[0]\Delta\varepsilon_{Swarm[0]}) = \$(Swarm[0])$;
  Then $localbest(Swarm[0])$, $mut(Swarm[0])$, $globalbest(Swarm[0])$、$upd(Swarm[0])$ are used to bring about the next generation $Swarm[1]$, follows by $loop(Swarm[1])$.

- The t-th iteration (t>1).

In the loop iteration, the basic cycle consists replaces an invisible $\varepsilon_{Swarm[t]}$ one step deep by one offspring $Swarm[t]$ and then $Swarm[t-1]$ forgotten. When a goal with the maximal reaches, i.e., the terminal condition is satisfied, then the procedure end with the current state as the best solution.

SIMULATION PLATFORM AND EXPERIMENT

System architecture: The whole architecture of the SimTool can be divided into 3 modules as shown in Fig. 3. The module I take responsibility for converting the service specification file (such as BPEL) into our

graph-based representation (APG) XML file and then transform it into the portable Pi-calculus agent; In nodule II, the XML of APG can be converted into the Pi-cost-calculus agent and then can form the AST file of each particle with the help of the code of it. In module IV, we integrate some existing tools for simulation and/or verification of Pi-calculus including bisimulation checkers, the Advanced Bisimulation Checker (ABC) (Briais, 2007) and the Mobility Workbench (MWB) (Victor and Moller, 1994) and the graphic simulator PiVizTool (Bog *et al*., 2007). In module III, we develop a simulator of DPSO for service composition based on open source project JSwarm-PSO (Pablo, 2009). As the algebraic representation for particles vary, the fitness functions are also different among particles. Our main changes based on it include: A functional module called Fitness Function Generator can provide the source file of ASTFitnessFunction to extend the abstract class FitnessFunction and can be reflected from the generated file; A class ParticleUpdateDiscrete to extend the abstract class ParticleUpdate based on the physical representation of particles, of which a new class Particle D replace the role of the class Particle. Although there exist some workbench or toolbox about Pi calculus, such as MWB and ABC, most of them have no any programmable interface, which limit the modeling and simulation with Pi calculus.

Taking this situation into account, we have developed an extended Pi compiler in Java, which attempts to be applied into the practice of modeling of service. For compatibility reasons, this compiler is not based on Pi-beam expressions directly, but on a two-stage translation: one is interpreting the Pi-beam expressions into portable Pi; the other is analyzing it to AST. AST play an important role in both modeling and QoS evaluation. The package syntax. AST is a basic of the whole compiler, consisting of process, type and value; another important package is syntax.error, which generalizes the syntax.syntax-error and syntax.typeerror. The front end of the compiler is composed of AST definition, parser, type checker, AST viewer and Swing interface. Lexical analyzer origins from a lexical analyzer generator, JFlex 1.3.5 (Klein *et al*., 2009), that can automatically bring out analyzer in Java according to the jflex lexicon specification. Syntax analysis module uses a parser generator, CUP v0. 10k (Hudson, 1999), to produce a syntax analyzer on the basis of LALR (1) grammar. The back end of the compiler is based on an assembly language on JVM, Jasmin (Jonathan *et al*. 2005), which is an assembler on JVM. It can translate the Java class into a sequence in form of instruction set of JVM. Finally, these assembler instructions could also be interpreted into byte file to run on JVM.

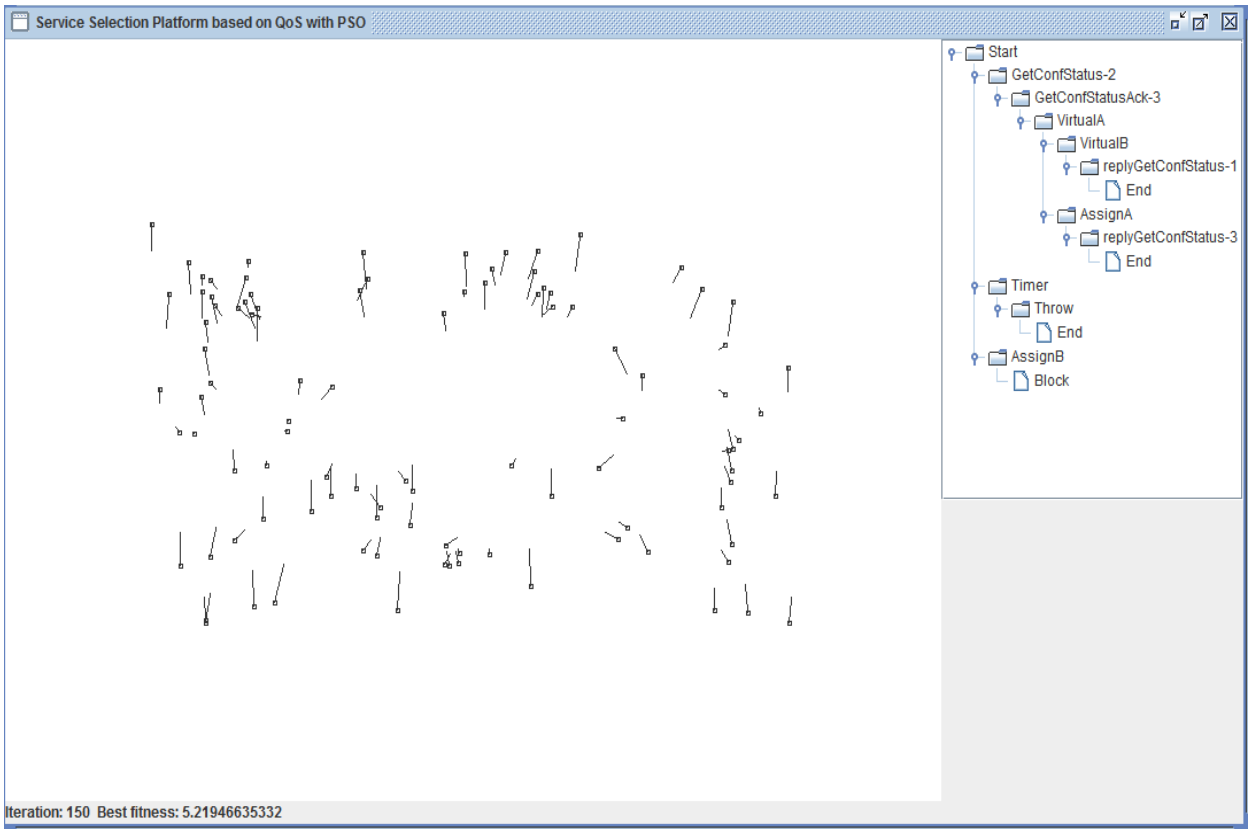

Fig. 4: Snapshot of particle swarm optimizer

EXPERIMENTAL RESULTS

Before the discussion of experiments, we first introduce how we get the QoS of services and the branch probability for fitness evaluation. Herein, we consider the executing time from "Start" to "End". For the internal activities such as assignments, we set breakpoint to obtain their average executing time; we distinguish 2 kind of channels: for the internal control channel we just get its QoS like the internal activities, but for the external message channel, we measure duration between the beginning and the end of them and merge the QoS of it into related activities. Moreover, the probabilities on each channel can also be compiled by statistic approaches. To test the simulator's performance and practical applicability, we have executed the following 2 experiments, on a computer with Intel(R) Petium(R) Dual Core 2.0GHz CPU, 4GB RAM, Windows XP 32bit, JDK 6u7 and Eclipse 3.3. The snapshot of the running interface is shown in Fig. 4.

- **Experiment 1:** Aggregated fitness evaluation in the different sizes of particles and iterations. Figure 5 exhibits the results of different numbers of particles in the DPSO. We can see a larger number of particles can obtain a better fitness value. A larger number of particles also converge fast to a larger fitness value in a smaller number of generations. According to the experimental result, it can be seen that the DPSO is able to select suitable candidate services for each composite service, if suitable number of particles is chosen.
- **Experiment 2:** Convergent generation evaluation in the different sizes of candidate services. This experiment observed the average convergent generations for different sizes of particles evaluated in different sizes of candidate services. Figure 6 clearly shows the result that the convergent generation increases with the increment of the size of candidate services no matter what particle number is. However, the increment ratio of convergent generation is lower than the increment ratio of the size of candidate services. For example, for particles = 40 case, the convergent generation is 277 when the size of candidate services is 100; when the size of candidate services is increased to 250, the convergent generation increases less than 8 generations only.

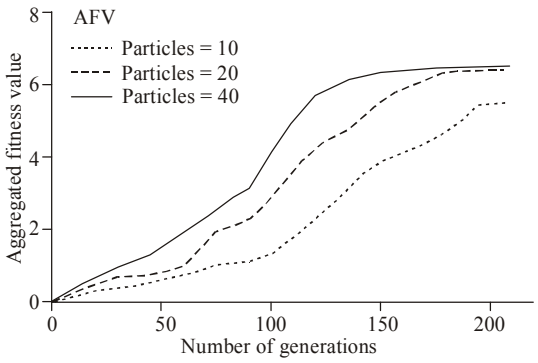

Fig. 5: Aggregated fitness evaluation in the different sizes of particles and iterations

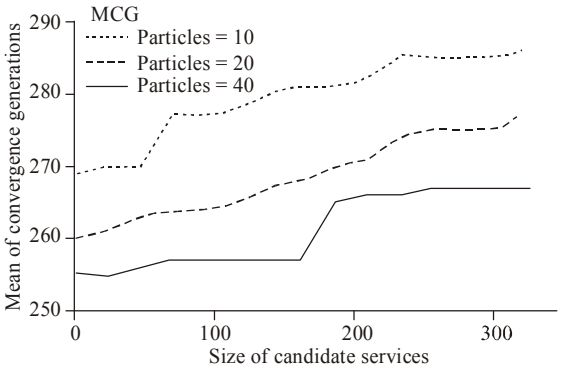

Fig. 6: Algorithmic flow schema of ECPSO convergent generation evaluation in the different of candidate services

CONCLUSION

In this study, we have proposed a new formal modeling of particle swarm optimizer from a perspective of a variant of Pi calculus, which treat the whole operations in PSO as a kind of meta-search procedure. On the basis of this mechanism, the QoS-aware service selection problem can be seen as a particular cost derivation under the LTS semantics. Moreover, a real simulator with a core of Pi calculus compiler is developed to verify our theory and also show the practical applicability.

ACKNOWLEDGMENT

This study is supported by the Talent Introduction Special Fund of Anhui Science and Technology University (Grant No. ZRC2011304).

REFERENCES

Bog, A., F. Puhlmann and M. Weske, 2007. The PiVizTool: Simulating Choreographies with Dynamic Binding. In: BPM of Demos, Vol. 272CEUR-WS.org, Retrieved from: http://www.bibsonomy.org/bibtex/2c14a7125d158f0980ec0a2fe97b3c4c2/dblp.

Briais, S., 2007. Advanced Bisimulation Checker. Retrieved from: http://lamp.epfl.ch/sbriais/abc/abc.html.

Eberbach, E. and P. Wegner, 2003. Beyond turing machines. Bull. EATCS, 81: 279-304.

Hudson, S.E., 1999. CUP Parser Generator for Java. Retrieved from: http://www.cs.princeton.edu/~ appel/modern/java/CUP/.

Jonathan, M. and R. Daniel, 2005. Jasmin. http://jasmin.source-forge .net/.

Kennedy, J. and R.C. Eberart, 1995. Particle swarm optimization. Proceedings of the IEEE International Conference on Neural Networks, 95: 1942-1948**.**

Kiniry, J.R., 1998. The specification of dynamic distributed component systems. M.Sc. Thesis, CS-TR-98-08, Computer Science Department, California Institute of Technology.

Klein, G., S. Rowe and R. Décamps, 2009. The Fast Scanner Generator for Java (JFlex). Retrieved from: http://jflex.de/.

Pablo, C., 2009. JSwarm-PSO. Retrieved from: http://jswarm-pso. sourceforge.net/.

Sumpter, D.J.T., G.B. Blanchard and D.S. Broomhead, 2001. Ants and agents: A process algebra approach to modelling ant colony behavior. Bull. Math. Biol., 63: 951-980.

Tofts, C., 1991. Describing social insect behaviour using process algebra. Trans. Soc. Comput. Simulat., 10: 227-283.

Victor, B. and F. Moller, 1994. The mobility workbench-a tool for the pi-calculus. Proceedings of the 6th International Conference on Computer Aided Verification, (CAV '94), Springer-Verlag London, pp: 428-440.