Using Combinatorial Particle Swarm Optimization to Automatic Service Identification

MOHAMED EL AMINE CHERGUI¹ and SIDI MOHAMED BENSILIMANE²
EEDIS Laboratory, Computer Science Department, Djillali Liabes University
B.P 89 Sidi Bel Abbès, 22000 Algeria
¹chergui.amine@gmail.com
²bensslimane@univ-sba.dz

Abstract: Service Oriented Architecture (SOA) has become one of the most suitable choices to ensure flexibility of the information system. But despite the consensus on the benefits of this type of architecture there are divergences in the approach to be followed for the development of this architecture. However, it is recognized that the success of a project based on service-oriented architecture requires proper services identification. Existing Service identification approaches are often prescriptive and based on the architect's experience thus could result in non-optimal designs which results in complicated dependencies between services. In this paper, an automated approach for identifying business services has been proposed by adopting several design metrics based on top-down decomposition of processes. This approach takes a business processes as input and produces a set of business services using a multi-objective combinatorial particle swarm optimization algorithm. The experimentation results denote that our approach achieves faster results and better performance.

Keywords—Service Identification; Combinatorial Particle Swarm Optimization; Service Oriented Architecture; Business Process Modeling.

1. INTRODUCTION:

Despite the current frenzy around SOA and the level of maturity reached by this architectural approach, there is no standard procedure for the establishment of such architecture.

Identification of services in service-oriented architecture is the first phase of the SOA lifecycle. It is considered the most important phase in a project to set up an SOA [4]. This phase not only determines the services that must be implemented, but also define the logic that must encapsulate each service. Proper identification of services to set up can avoid significant costs to the company and allows it to take full advantage of the SOA concept. The identification of services can be made from a variety of sources. This diversity of sources can mislead analysts SOA. The challenge is to develop an approach that best assists analysts to systematically search and analyze the vast amount of information that must be faced.

The need for an approach to identify services is recognized by several authors, who also agreed that services should be defined based on business process of the organization. Their work, however, does not present detailed analysis of the activities and services identification methods often provides principles and guidelines that are very difficult to follow in practice due to the lack of a systematic process.

The service identification is a multi-objective optimization problem. On the one hand, these methods have to make a compromise between different principles of SOA. On the other hand, the corresponding technical measures are not quantifiable, where the validity of decisions depends on the assessment of the architect.

In this paper, a new identification method is proposed, which aims to solve the above problems by supporting automation capabilities, by adopting technical measures, and using business process modeling. This process generates candidate software services using multi-objective combinatorial particle swarm optimization algorithm that analyses dependencies between business activities and business entity in order to group them into distinct services represented as clusters.

The remaining part of the paper is organized as follows: In section 2, the most related work is briefly reviewed. Section 3 introduces the particle swarm optimization. In section 4 we present the proposed service identification approach that uses Combinatorial Particle Swarm Optimization. Section 5 provide an implementation and experiment results to demonstrate the performance of our approach. Finally, section 6 concludes the paper and outlines directions for future work.

2. RELATED WORK

The literature provides a lot of work in service identification approaches, ranging from top-down to bottom-up. In this section, we briefly review the most related work in service identification.

Kazemi et al [9] have presented an automated method for identifying business services by adopting design metrics based on top-down decomposition of processes. This method takes a set of enterprise business processes as input and produces a set of non-dominated solutions representing appropriate business services using a multi-objective genetic algorithm.

Azevedo et al [2] proposed a top-down approach for services identification from business process models, applying heuristics to define services from the semantic analysis of process elements such as business rules and business requirements, and from a syntactic analysis of process models according to its corresponding structural patterns.
Kang et al. [8] presented a method of service identification using ontology for product line. Primary, Semantic relationship was derived through the mapping be-tween feature modeling and ontology. Second, both service and service boundary was defined by semantic distance. Third, the method was proposed for feature grouping and candidate service refining service candidate which is the fittest service granularity.

Suntae et al. [14] introduced a service identification method based on scenario modeling and a conceptual framework to elicit possible business changes. Traceability among business requirements, business changes and the identified services are also supported by their method.

Rana et al. [12] have introduced a generic ontology-based framework, BPAOnto-SOA, for the derivation of software service oriented models from a given business process architecture relying on two ontologies. This framework utilizes an adapted clustering algorithm based on affinity analysis of business process functions in order to group them into services along with their associated NFRs to ensure conformance of these services with SOA principles.

Inaganti et al. [5] mentions some best practices, wherever appropriate, to point out the vagueness involved in service identification. A top-down and bottom-up technique for service identification has proposed in this methodology.

Jamshidi et al. [6] present a novel approach called ASIM for automatically identifying and partly specifying enterprise-level software services from business models using best practices and principles of model-driven software development. They formulated service identification as a multi-objective optimization problem and solved it by a novel meta-heuristic optimization algorithm that derives appropriate service abstractions by using appropriate quantitative measures for granularity, coupling, cohesion, reusability, and maintainability.

Soltani et al. [13] introduce a meta-heuristic approach called MOOSI for deriving service-oriented architectures from annotated business process model. They generate candidate software services using multi-objective evolutionary algorithm that analyses dependencies between business activities in order to group them into distinct clusters, each cluster must groups one or more closely related activities to form a future software service.

3. PARTICLE SWARM OPTIMIZATION

Particle Swarm Optimization (PSO), firstly introduced by Kennedy and Eberhart [10] is one of the most recent metaheuristics, which is inspired by the swarming behavior of animals and human social behavior. They developed this method for optimization of continuous non-linear functions. The algorithm is similar to other population-based algorithms like Genetic algorithms but, there is no direct combination of individuals of the population. Instead, it relies on the social behavior of the particles. In every generation; each particle adjusts its trajectory based on its best position (local best) and the position of the best particle (Global best) of the entire population. This concept increases the stochastic nature of the particle and converge quickly to a global minimum with a reasonable good solution.

3.1 PSO Algorithm

The general principles of the PSO algorithm are stated as follows. Similarly to an evolutionary computation technique, PSO maintains a population of particles, where each particle represents a potential solution to an optimization problem [3].

Let \( m \) be the size of the swarm. Each particle \( i \) can be represented as an object with several characteristics. Suppose that the search space is a \( n \)-dimensional space, and then the \( i \)th particle can be represented by a \( n \)-dimensional vector \( \mathbf{x}_i = [x_{i1}, x_{i2}, ..., x_{in}] \), and velocity \( \mathbf{v}_i = [v_{i1}, v_{i2}, ..., v_{in}] \), where \( i = 1, 2, ..., m \).

In PSO, particle \( i \) remembers the best position it visited so far, referred as \( P_i = (p_{i1}, p_{i2}, ..., p_{in}) \), and the best position in the swarm, referred as \( G = (G_1, G_2, ..., G_n) \).

PSO is similar to an evolutionary computation algorithm and, in each generation, particle \( i \) adjusts its velocity \( v_{ij}^t \) and position \( x_{ij}^t \) for each dimension \( j \) by referring to, with random multipliers, the personal best position \( p_{ij}^t \) and the swarm’s best position \( G_j^t \), using Eqs. (1) and (2), as follows:

\[
v_{ij}^t = v_{ij}^{t-1} + c_1 r_1 (p_{ij}^{t-1} - x_{ij}^{t-1}) + c_2 r_2 (G_j^{t-1} - x_{ij}^{t-1}) \tag{1}
\]

\[
x_{ij}^t = x_{ij}^{t-1} + v_{ij}^t \tag{2}
\]

Where \( c_1 \) and \( c_2 \) are the acceleration constants and \( r_1 \) and \( r_2 \) are random real numbers drawn from \([0, 1]\). Thus the particle flies through potential solutions toward \( P_i \) and \( G^t \) while still exploring new areas. Such stochastic mechanism may allow escaping from local optima. Since there was no actual mechanism for controlling the velocity of a particle, it was necessary to impose a maximum value \( V_{max} \) on it. If the velocity exceeded this threshold, it was set equal to \( V_{max} \), which controls the maximum travel distance at each iteration, to avoid a particle flying past good solutions. The PSO algorithm is terminated with a maximum number of generations or the best particle position of the entire swarm cannot be improved further after a sufficiently large number of generations.

The aforementioned problem was addressed by incorporating a weight parameter in the previous velocity of the particle. Thus, in the latest versions of the PSO, Eqs. (2) and (3) are changed into the following ones:

\[
v_{ij}^t = \chi (v_{ij}^{t-1} + c_1 r_1 (p_{ij}^{t-1} - x_{ij}^{t-1}) + c_2 r_2 (G_j^{t-1} - x_{ij}^{t-1})) \tag{3}
\]

\[
x_{ij}^t = x_{ij}^{t-1} + v_{ij}^t \tag{4}
\]
\( \omega \) is called inertia weight and is employed to control the impact of the previous history of velocities on the current one. Accordingly, the parameter \( \omega \) regulates the trade-off between the global and local exploration abilities of the swarm. A large inertia weight facilitates global exploration, while a small one tends to facilitate local exploration. A suitable value for the inertia weight \( \omega \) usually provides balance between global and local exploration abilities and consequently results in a reduction of the number of iterations required to locate the optimum solution. \( \chi \) is a constriction factor, which is used to limit the velocity.

The PSO algorithm has shown its robustness and efficacy for solving function value optimization problems in real number spaces. Only a few researches have been conducted for extending PSO to combinatorial optimization problems.

### 3.2 PSO for combinatorial optimization

Jarboui et al [7] introduced CPSO (Combinatorial particle swarm optimization) for solving the combinatorial optimization problem. Combinatorial PSO essentially differs from the original (or continuous) PSO in some characteristics.

1) **Definition of a particle**

Denote by \( Y_t = (y_{1t}, y_{2t}, ..., y_{nt}) \) the n-dimensional vector associated to the solution \( X_t = (x_{1t}, x_{2t}, ..., x_{nt}) \) taking a value in \((-1,0,1)\) according to the state of solution of the \( t \)-th particle at iteration \( t \).

\( y_i \) is a dummy variable used to permit the transition from the combinatorial state to the continuous state and vice versa.

\[
y_i^t = \begin{cases} 
1 & \text{if } x_i^t = G_i^t, \\
-1 & \text{if } x_i^t = p_{ij}^t, \\
-1 \text{ or } 1 \text{ randomly if } (x_i^t = G_i^t = p_{ij}^t), \\
0 & \text{otherwise.}
\end{cases}
\]

2) **Velocity**

Let \( d_1 = -1 - y_i^{t-1} \) be the distance between \( y_i^{t-1} \) and the best solution obtained by \( t \)-th particle.

Let \( d_2 = 1 - y_i^{t-1} \) be the distance between the current solution \( x_i^{t-1} \) and the best solution obtained in the swarm.

The updated equation for the velocity term used in CPSO is then:

\[
v_i^t = w_v v_i^{t-1} + r_1 c_1 d_1 + r_2 c_2 d_2, \\
v_i^t = w_v v_i^{t-1} + r_1 c_1 (-1 - y_i^{t-1}) + r_2 c_2 (1 - y_i^{t-1}).
\]

With this function, the change of the velocity \( v_i \) depends on the result of \( y_i^{t-1} \).

3) **Construction of a particle solution**

The updated equation of the solution is computed within \( y_i^{t} \):

\[
\lambda_i^{t} = y_i^{t-1} + v_i^t
\]

The value of \( y_i^{t} \) is adjusted according to the following function:

\[
y_i^{t} = \begin{cases} 
1 & \text{if } 10d_i^{t} > \alpha \\
-1 & \text{if } 10d_i^{t} < -\alpha \\
0 & \text{otherwise}
\end{cases}
\]

The new solution is

\[
x_i^{t} = \begin{cases} 
G_i^{t-1} & \text{if } y_i^{t} = 1, \\
p_i^{t-1} & \text{if } y_i^{t} = -1, \\
a \text{ random number otherwise}
\end{cases}
\]

### 4. The Proposed Approach

Referring to the SOMA [1], different approaches can be adopted to identify services, namely top-down (domain decomposition), bottom-up (existing system analysis) and meet-in-the-middle. The Top-Down approach consists mainly in decomposing business processes into finer business tasks. The bottom-up approach is about analyzing existing IT assets and finding functionality that could be exposed as services. The last one is about conducting both a bottom-up analysis and top-down analysis and to correlate the services identified by each of these approaches.

In our approaches, we begin with business process model as input (Top-Down) and derive the effective service set based on this model. We represent the business process model with CRUD matrix [6].

#### 4.2 CRUD Matrix

A matrix represents the impact of business activity on business entities. This structure is more appropriate to facilitate automated software engineering activities, which has EBP's as its rows, BEs as its columns, also semantic relationships ("C", "U", "D", "R"(Create, Update, Delete, Read), with the distinct intensity (relationship strength) C>D>R as its cells.

In order to compute the value of each technical metrics, the model should be transformed into value-based rather than tag-based. Therefore, the corresponding value (weight) of each tag should be replaced in the model (see Fig. 1) according to 1=0>C=U>D>R=0.1.

Also due to simplifying computations and have rounded values, we adopt these substitutions: C: =1, U: =0.75, D: =0.5, R: =0.25.

**Definition 1 (Business Entity):** A Business Entity (BE) can be defined as BE = \{n, A, R\}.

Where \( n \) is the name of the business objects, \( A \) is the set of attributes, and \( R \) is the set of relationship between BE and other business entities.

**Definition 2 (Elementary Business Process):** An Elementary Business Process (EBP) can be defined as EBP = \{n, (BE, sr)\}.

Where \( n \) is the name of the elementary business process,
BE_{ij} is the jth business entity which semantically related to corresponding EBP. sr ∈ (C, R, U, D) is the type of semantic relationship between EBP and BE_{ij}; “C” means the EBP Creates the BE_{ij}, “R” means the EBP Reads the BE_{ij}, “U” means the EBP Updates the BE_{ij}, “D” means the EBP Deletes the BE_{ij}.

It is intuitively obvious that each of the semantic Relationships makes dissimilar intensity between an EBP and a BE. For instance, creation relationship makes the most intensity between an EBP and a BE, but the read relationship makes the least intensity between them.

It is intuitively obvious that each of the semantic Relationships makes dissimilar intensity between an EBP and a BE. For instance, creation relationship makes the most intensity between an EBP and a BE, but the read relationship makes the least intensity between them.

**Definition 3 (Cluster):** The kth cluster of the CRUD matrix (Fig 1) can be defined as: Cluster\(_k\) = \{ (EBP\(_i\), BE\(_j\)), i = l1...h1, j = l2...h2, Where 1 ≤ l1<h1≤#row and 1 ≤ l2 < h2 ≤ #column \}

**Definition 4 (Characteristics of the clusters):** If Cluster\(_k\) = \{ (EBP\(_i\), BE\(_j\)), i = l1 ... h1, j = l2 ... h2 \} , Cluster\(_k\) = \{ (EBP\(_i\), BE\(_j\)), i = l’ ... h’, j = l’ ... h’2 \} , M as a CRUD matrix and S as software service, then the following statements as the characteristics of the matrix are going to be correct.

- Non-intersection: ∀ k, k’Cluster\(_k\) ∩ Cluster\(_k’\) = ∅ - It means that the clusters have nothing in common in its structural and behavioral elements.
- ∀ k, l’ ∈ [l1, l2], then Cluster\(_k\) U Cluster\(_k’\) = \{ (EBP\(_i\), BE\(_j\)), i = l1 ... h’, j = l2 ... h’2 \} – It means that the union of two clusters is equivalent to merging them.

- Completeness: If M is clustered into n clusters then ∪\(_{k=1}^{n}\) Cluster\(_k\) = M - Union of all the clusters constituting the CRUD matrix is equivalent to that matrix.
- ∀ S ∃ k Cluster\(_k\) = S – There is exactly one cluster, which is equivalent to the specified service S.

### 4.3 Technical Metrics

In order to automate the service identification process, appropriate metrics should be formulated with the elements of the CRUD matrix, which were defined in section 3. Each metric should introduce a function, which determines the appropriate value of the corresponding metric. Jamshidi [6] has defined four metrics, including total semantic relationship, internal semantic dependency, external semantic dependency and service semantic affinity. For each metric, authors at the first define a function to evaluate a service \(S_k\) and then define a function to evaluate a service set \(S\) constituting \(S_k\).

1) **Metric 1: Total Semantic Relationship (TSR)**

TSR for a service is defined by summation of total semantic relationships that are belonging to the service cluster \(S_k\). The TSR formula, TSR (\(S_k\)), is considering the number and type of semantic relationships associated with the \(S_k\) cluster to measure its size. For a service set \(S\) total semantic relationships, TSR (\(S\)), is defined as the average of services TSRs. TSR (\(S_k\)) and TSR (\(S\)) are formulated as:

\[
TSR(S_k) = \sum_{i=1}^{h2} \sum_{j=1}^{h1} sr_{ij}
\]

\[
TSR(S) = \frac{\sum_{k=1}^{#s} TSR(S_k)}{#s}
\]

Where TSR (\(S_k\)) is the total semantic relationship of a service\(S_k\). TSR (\(S\)) is the total semantic relationship of service set \(S\). \(S_k\) is \(k\)th service in service set \(S\). Service \(S_k\) is located in a cluster with the boundaries \(i=1...h1, j=1...h2\). \(sr_{ij}\) is the value of semantic relationship between EBP\(_i\) and BE\(_j\) which specified by M (\(i,j\)). #s is the total number of services in service set \(S\).

2) **Metric 2: Internal Semantic Dependency (ISD)**

Internal semantic dependency of a service is defined by degree of strength of relationship between service semantic relationships within a service. Within CRUD matrix, each semantic relationship is strongly related to other semantic relationships, which are located in the same elementary business process and which are related to one business entity.

\[
ISD(S_k) = TSSD(S_k)/TSSD(S_k)
\]

\[
ISD(S) = \frac{\sum_{k=1}^{#s} ISD(S_k) * TSR(S_k)}{\sum_{j=1}^{#row} \sum_{i=1}^{#col} sr_{ij}}
\]
Metric 3: External Semantic Dependency ESD

External semantic dependency is defined as a measure of the relative interdependence among service semantic relationships (semantic relationship in the service cluster) and external semantic relationships (semantic relationships which are not belong to service cluster). Within CRUD matrix, each cell $s_{ij}$, which is not categorized in any cluster, represents an external dependency.

$$\text{ESD}(S_k) = \sum_{k=1}^{\#s} \text{ESD}(S_k) = \left( \sum_{j=1}^{\#col} \sum_{i=1}^{\#row} s_{ij} \right) - \sum_{k=1}^{\#s} \text{TSR}(S_k)$$

$$\text{ESD}(S) = \sum_{k=1}^{\#s} \text{ESD}(S_k) = \left( \sum_{j=1}^{\#col} \sum_{i=1}^{\#row} s_{ij} \right) - \sum_{k=1}^{\#s} \text{TSR}(S_k)$$

Metric 4: Service Semantic Affinity (SSA)

Service semantic affinity of a service $S$, $SSA(S_k)$, is defined by density of the semantic relationships in the service cluster. Furthermore, we define affinity of the service set $S$, $SSA(S)$, as the average of density of service’s clusters.

$$SSA(S_k) = \sum_{j=1}^{\#col} \sum_{i=1}^{\#row} s_{ij} / (h^2 - 12 + 1) \ast (h1 + l1 + 1)$$

$$SSA(S) = \sum_{k=1}^{\#s} \text{TSR}(S_k) / (h^2 - 12 + 1) \ast (h1 - h1 + 1)$$

Discussion on metrics

Jamshidi et al. [6] introduced four metrics (total semantic relationship, internal semantic dependency, external semantic dependency and service semantic affinity) as criteria for identifying and evaluating services.

Table 1 shows the summary for proposed metrics. These metrics are introduced to guide the service identification process to shape high quality services to attain high cohesion, low coupling, course grained granularity, high maintainability and high reusability design principles. Table 2 shows the relationship between metrics and technical goals.

4.4 Resolving Clustering Optimization Problem

Matrix clustering is considered as NP-complete problem, we have to use Meta-heuristics to solving it. We propose Combinatorial Particle Swarm Optimization [7] for clustering the CRUD matrix.

1) Solution representation

One of the most important issues when designing the PSO algorithm lies on its solution representation. We setup search space of $n$-dimension for $n$-objects. Each dimension represents an object and particle $X_i = \{x_{1i}^1, x_{2i}^1, ..., x_{ni}^1\}$ corresponds to affection of $n$ objects, such that $x_{ij}^k \in \{1, 2, ..., k\}$, where $k$ is the number of classes as in [11]. The length of $X_i$ represent the number of rows plus the number of columns of the CRUD Matrix.

The scheme of Fig. 2 illustrates the solution representation of particle $X_i$ of CPSO algorithm.

$$\begin{align*}
\text{EBP}_1 & \quad \text{EBP}_2 \quad \cdots \quad \text{EBP}_n \\
\{1, 2, \ldots, k\} & \quad \{1, 2, \ldots, k\} \quad \cdots \quad \{1, 2, \ldots, k\} \quad \{1, 2, \ldots, k\} \quad \cdots \quad \{1, 2, \ldots, k\}
\end{align*}$$

Fig.1. Solution representation

Table 1. Metrics summary

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Main Purpose</th>
<th>Optimization goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSR</td>
<td>Services which comprise more semantic relationships</td>
<td>Max</td>
</tr>
<tr>
<td>ISD</td>
<td>Services with more internal dependency</td>
<td>Max</td>
</tr>
<tr>
<td>ESD</td>
<td>Low semantic dependency between services</td>
<td>Min</td>
</tr>
<tr>
<td>SSA</td>
<td>services deals with a little number of EBP's and BEs</td>
<td>Max</td>
</tr>
</tbody>
</table>

Table 2. Relationship between metrics and technical goals

<table>
<thead>
<tr>
<th>Service Design Principle</th>
<th>Meanings</th>
<th>Optimization Direction</th>
<th>Correlated Metrics</th>
<th>Opposed Metrics</th>
<th>Assisted Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cohesion</td>
<td>Semantic closeness between operations in $S$</td>
<td>High</td>
<td>ISD</td>
<td>ESD, TSR</td>
<td>SSA</td>
</tr>
<tr>
<td>Coupling</td>
<td>Semantic closeness between operations in $S$ and in other services</td>
<td>Low</td>
<td>ESD</td>
<td>ISD</td>
<td>SSA</td>
</tr>
<tr>
<td>Granularity</td>
<td>The scale of $S$</td>
<td>Course grained</td>
<td>TSR</td>
<td>ISD</td>
<td>SSA, ESD</td>
</tr>
<tr>
<td>Maintainability</td>
<td>Ease of modifying $S$ to fit proposed requirements</td>
<td>High</td>
<td>ESD and ISD</td>
<td>TSR</td>
<td>SSA</td>
</tr>
<tr>
<td>Reusability</td>
<td>Capability of adoption of $S$ in the other contexts</td>
<td>High</td>
<td>ESD and ISD</td>
<td>TSR</td>
<td>SSA</td>
</tr>
</tbody>
</table>
2) The Objective Function

Four metrics TSR(S), ISD(S), ESD(S), and SSA(S) are defined. There are conflicts between these objectives. We reformulate the problem to a single objective problem as in Jamshidi et al [6]:

\[ Z(S) = \text{F}(\text{TSR}(S), \text{ISD}(S), \text{EST}(S), \text{SSA}(S)) \]
\[ Z(S) = \text{ATSR}(S) \times \text{ISD}(S) \times (1 - \text{AESD}(S)) \times \text{SSA}(S) \] (11)

Where:
\[ \text{ATSR}(S) = \alpha \times \text{TSR}(S) \]
\[ \text{AESD}(S) = (1 - \alpha \times \text{ESD}(S)) \]
\[ \alpha = \frac{1}{\sum_{i=0}^{\text{#row}} \sum_{j=0}^{\text{#col}} s_{ij}} \]

3) The CPSO-based Clustering Algorithm

The CPSO algorithm is similar to other evolutionary algorithms. In CPSO, the population is the number of particles in a problem space. Particles are initialised randomly. Each particle will have a fitness value, which will be evaluated by a fitness function to be optimised in each generation. We illustrate in the following the use of a clustering strategy based on CPSO algorithm.

**Algorithm : CPSO-based clustering algorithm**

**Input:** CRUD Matrix  
**Output:** a set of clusters  
**Step 1:** Set particle dimension as equal to the number of rows plus the number of columns of the CRUD Matrix.  
**Step 2:** Initialize the particles position \( X_i \) and velocity \( v_i \) randomly, and set the maximum number of iterations.  
**Step 3:** For each particle, calculate its fitness value according to Equation (11)  
**Step 4:** If the fitness value is better than the previous best pbest, set the current fitness value as the new pbest.  
**Step 5:** After Step 3 and 4 for all particles, select the best particle as gbest  
**Step 6:** For all particles, calculate velocity using Equation (7) and update their positions using Equation (10)  
**Step 7:** If the stopping criteria or maximum iteration is not satisfied, repeat from step 3.

5. IMPLEMENTATION AND EXPERIMENTAL RESULTS

We have implemented our approach using C# with Microsoft visual studio 2010. All experiments run in Windows 7 on desktop PC with Intel Dual Core, 2.3 GHz processors.

The CPSO parameters were set experimentally as follows. Inertia weight takes a value of \( w = 0.4091 \). Parameters \( c_1 \) and \( c_2 \) were set to 2.1304 and 1.0575 respectively. The swarm comprises of 156 particles and 5000 iterations of CPSO algorithm were used according to Magnus [11].

We represent CRUD matrix as XML file. After treatment of the matrix, we display a graph showing the evolution of the fitness and the identified services. Fig 3 and 4.

![Fig.2. Graph of Fitness](image)

![Fig.3. Identified Services](image)

To evaluate the performance of our tool SIPSO (Service Identification Particle Swarm Optimization), we applied the proposed clustering algorithm on four business processes. The provided results are shown in Table 3. The optimal solution is found in a reduced number of iterations.

<table>
<thead>
<tr>
<th>Business Process</th>
<th>BP1</th>
<th>BP2</th>
<th>BP3</th>
<th>BP4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of EBP</td>
<td>8</td>
<td>11</td>
<td>9</td>
<td>13</td>
</tr>
<tr>
<td>Number of BE</td>
<td>6</td>
<td>9</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>Number of Clusters</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Fitness</td>
<td>0.0938</td>
<td>0.0439</td>
<td>0.0587</td>
<td>0.0428</td>
</tr>
<tr>
<td>Found at iteration</td>
<td>14</td>
<td>544</td>
<td>243</td>
<td>1221</td>
</tr>
</tbody>
</table>

As shown in Table 4, the number of clusters and their composition varies according to the selected metrics. Experimentations have shown that the best results are obtained by combining the four metrics: Total Semantic Relationship, Internal Semantic Dependency, External Semantic Dependency and Service Semantic Affinity.
Table 4. Experimental Results by Combining Differences Metrics

<table>
<thead>
<tr>
<th>Metrics</th>
<th>Features</th>
<th>Number of Clusters</th>
<th>Fitness</th>
<th>Found at iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using four metrics</td>
<td>TSR,ISD,ESD,SSA</td>
<td>4</td>
<td>0.0428</td>
<td>1221</td>
</tr>
<tr>
<td>Using three metrics</td>
<td>TSR,ISD,ESD</td>
<td>2</td>
<td>0.1367</td>
<td>1100</td>
</tr>
<tr>
<td>Using two metrics</td>
<td>TSR,ISD,ssa</td>
<td>4</td>
<td>0.0495</td>
<td>1239</td>
</tr>
<tr>
<td>Using one metric</td>
<td>ISD,ESD,SSA</td>
<td>4</td>
<td>0.1269</td>
<td>1408</td>
</tr>
</tbody>
</table>

We run our tool on the same business process used in Jamshidi et al [6]. We note, as shown in Table 5, that we have wildly improved (reduced) the number of iterations to achieve the same results as in Jamshidi et al [6].

Table 5. SIPSO Compared to ASIM [6]

<table>
<thead>
<tr>
<th>Features</th>
<th>ASIM [8]</th>
<th>SIPSO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of EBP</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>Number of BE</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Number of Clusters</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Fitness</td>
<td>0.0428</td>
<td>0.0428</td>
</tr>
<tr>
<td>Found at iteration</td>
<td>5312</td>
<td>1221</td>
</tr>
</tbody>
</table>

6. CONCLUSION

Among the steps of a service life-cycle model, we focused on the services identification step. We propose a top-down approach for services identification from business process models using clustering CPSO-based solution, to automatically identify the services using the appropriate metrics.

The experimentation results show that our approach achieves high performance in term of convergence speed compared with other solution.

As future work, we are studying introducing others metrics and semantic annotation to fully considering semantic relationships between business elements, and hence transaction and semantic integrity can be guaranteed. Our approach also can be improved while using hybrid-clustering algorithm by embedding the crossover and mutation operators of Genetic Algorithm into the CPSO in order to improve the local search ability and to enhance performance.

REFERENCES
