

Measures of Structural Complexity and Service Autonomy

A. Rostampour*, A. Kazemi*, F. Shams*, P. Jamshidi*, A. Nasirzadeh Azizkandi*

*Automated Software Engineering Research Group

Electrical and Computer Engineering Faculty, Shahid Beheshti University GC,
Tehran, Iran

E-mail : { A.Rostampour, Ali.Kazemi }@mail.sbu.ac.ir, { F.Shams, P.Jamshidi }@sbu.ac.ir,
Ali_Nasirzadeh@tabrizu.ac.ir

Abstract—Analytical approaches based on rigorous mathematical foundation are getting more importance in service-oriented computing (SOC) research area. Defining appropriate metrics to measure service quality attributes is one of the key activities in this context which plays an important role in construction of quality service-oriented solutions. In this paper, some metrics are introduced for measuring structural complexity and service autonomy in service-oriented modeling phase which have a great impact on other quality attributes such as maintainability and reusability. Finally we describe the presented metrics in details, using an example, and evaluate them Theoretically and empirically.

Keywords- Service quality attributes; structural complexity; autonomy; measurement; CRUD matrix

I. INTRODUCTION

Today, service oriented architecture is one of the modern approaches in building distributed systems and has found its place in organizing information systems especially web-based systems [1]. As independent software entities, services are fundamental constructive components of SOA. Achieving a right level of quality attributes such as Structural complexity, Reusability, Maintainability and Performance, is amongst the most important issues in software development. Structural complexity and autonomy are two important quality attributes in SOA which have a great impact on other attributes considered for a service. Structural complexity can affect reusability, performance and Maintainability [2], and autonomy has a great impact on other attributes such as discoverability, reusability and statelessness of service.

Lots of work have been done on measuring the structural complexity of Software, Workflow and Component [2],[3],[4],[5], but the study on service structural complexity, especially in the service modeling phase, is limited to a few [1],[6],[7],[8]. Most of these studies have measured the structural complexity only from a specific point of view and some have proposed metrics that could not be measured in an automated manner [1], [7]. No metric to the knowledge of the authors, has been provided to measure service autonomy in Service-oriented modeling phase.

After identifying the problem, we have set the following objective for the research reported in the present paper:

“To propose some metrics for measuring Structural Complexity and Service Autonomy in service oriented modeling phase”

The output of service-oriented modeling phase is the basis for the implementation phase [9], thus by measuring service structural complexity and autonomy in modeling phase, high quality services could be achieved and the cost of implementation could be decreased substantially.

One of the important applications of measuring structural complexity and autonomy is service identification with respect to quality attributes, and as a criterion in evaluation of identified candidate services.

In this paper we define service as a set of Elementary Business Process (EBPs) which perform operations such as Create ,Read, Update and Delete on Business Entities (BEs). These services are represented in the form of clusters on CRUD Matrix [10],[11],[12],[13],[14]. Based on this definition, autonomy and structural complexity of each service will be calculated.

II. BASIC CONCEPTS

In this section, the definitions of relevant terminology and its implications are introduced. Note that among researchers there is no universal agreement as to the precise definitions of the following terms. The reader is encouraged to view these as a consistent set of terminology and indicative of what it is meant by many researchers and practitioners in the field. In order to define the terminologies precisely, the definitions are introduced by lightweight formal elements [14].

Definition 1: A Business Entity (BE) is a triple like $BE = (n, A, R)$, where n is the name of business, A is a set of attributes and R is the relations between BE and other business entities [14].

Definition 2: An Elementary Business Process (EBP) is defined as a set of pairs like $EBP_j = \{ (BE_i, SR) \}$ where BE_i is the i th business entity which has a relation with EBP_j by other conceptual relationships $SR = \{ "C", "R", "U", "D" \}$ [15].

Definition 3: A CRUD matrix is defined as $M_{row \times column}$ where Row is the number of rows and Column is the number of columns of matrix. The set of EBPs is placed in the rows of this matrix and the set of BEs is placed in the columns of this matrix [14].

Definition 4: The k 'th cluster of the CRUD matrix can be defined as Cluster $k = \{ (EBP_i, BE_j), i=1...h1, j=1...h2 \}$, where $1 \leq h1 \leq \#row$ and $1 \leq h2 \leq \#column$ [14].

Definition 5: A software service can be defined as $S = \{n, I, \text{Msg}, \text{RS}\}$, where n is the name of service, I is the set of operations associated with S , Msg is the set of messages corresponding to S , and RS is the set of relationship between S and the other services in the service model. Referring to the CRUD matrix, $I \subset \text{EBP}$ is the set of corresponding EBPs, and $\text{Msg} \subset B$ is the set of corresponding BEs [14].

Referring to the definition 5, each cluster has some characteristics as follows.

Definition 6: (Characteristics of the clusters):

If Cluster $k = \{(\text{EBP}_i, \text{BE}_j), i=1 \dots h_1, j=1 \dots h_2\}$, Cluster $k' = \{(\text{EBP}_i, \text{BE}_j), i=1' \dots h'_1, j=1' \dots h'_2\}$, M as a CRUD matrix, and S as a software service [14]:

- Non-intersection: $\forall k, k' \text{ Cluster } k \cap \text{Cluster } k' = \{\}$ – It means that the clusters have nothing in common in its structural and behavioral elements.
- $\forall k, k'$ let $1' = h_1, 1' = h_2$, then $\text{Cluster } k \cup \text{Cluster } k' = \{(\text{EBP}_i, \text{BE}_j), i=1 \dots h'_1, j=1' \dots h'_2\}$ – It means that the union of two clusters is equivalent to merging them.
- Completeness: If M is clustered into n clusters then $\bigcup_{k=1}^n \text{Cluster}_k = m$ – Union of all the clusters constituting the CRUD matrix is equivalent to that matrix.
- $\forall S \exists k \text{ Cluster } k = S$ – There is exactly one cluster, which is equivalent to the specified Service S .

III. SERVICE STRUCTURAL COMPLEXITY

In the design level, structural complexity of a service could be defined based on its interface; the simpler the interface, the less complex the service is. Therefore structural complexity of a service could be defined based on its operations and messages.

As stated earlier, a service is defined as a cluster on the CRUD matrix. In order to measure structural complexity, we must calculate structural complexity of messages represented by the BEs in the CRUD matrix.

A BE is defined as $\text{BE} = \{n, A, R\}$; where n is the business entity name, A is a set of attributes and R is the BE's set of relationships. Considering this definition, the structural complexity of BE could be determined based on its attributes.

Metric1: Business Entity (BE) complexity

Complexity of a BE is defined as the sum of the complexities of its attributes. We classify the attributes of a BE into two categories; Primitive and Composite. Also we classify the composite type into Simple and Complex categories. Then, as shown in Table I, in accordance with different structures, the value of complexity for each category is assigned empirically.

TABLE I. DATA TYPE CLASSIFICATION AND STRUCTURAL COMPLEXITY VALUE

Data Type	Example	complexity
Primitive	Integer, Floating point	1
Simple Composite	Date, String	5
Complex Composite	Class	10

Metric2: Service Interface complexity

Service specifications are derived from the clusters in the CRUD matrix that represents the services. In each cluster, EBPs which deal with at least one BE, are interpreted as

operations of the service represented by that cluster. The input and output messages of an operation are the BEs that are manipulated through one or more of the actions: "Creation", "Reading", "Updating" and "Deletion" [14].

In order to calculate the interface complexity of a service, the complexity of each EBP is calculated according to the complexity of BEs, discussed earlier.

We define structural complexity of an EBP based on the summation of its input and output BEs' complexity.

Formula (1) calculates the complexity of EBP_j of i th service:

$$\text{EBP}_j C = \sum_{k=L_{2i}}^{h_{2i}} x_{jk} \times M[0, k]. \quad (1)$$

where:

i : is service number

j : is EBP number

L_{2i}, h_{2i} : bound of existing BEs in the i th service

$M[0, k]$: structural complexity corresponding to BE_k in the row 0 and column k of CRUD matrix

$x_{jk} = 1$: if jk th element of CRUD matrix is C, R or D since in these three cases we have one message which is either input or output [15].

In *Creation* of a BE by an EBP, an output message by the service is produced through the corresponding operation.

Reading a BE by an EBP represents an input message to the operation of the service.

Updating a BE by an EBP includes two messages: one input message to the service, and the other is the updated information as an output message.

Deletion of a BE by an EBP requires an input message to the service and has no output message.

$x_{jk} = 2$: if jk th element of CRUD matrix is U. Clearly the update structural complexity is more than other actions' structural complexity because in the case of an update, a BE is defined as the input of update action, and the modified version of the BE is the output in service operation.

Finally, interface complexity of a service is calculated based on the sum of complexities of EBPs provided by that interface.

Formula (2) calculates the interface complexity of i th service.

$$S_i C = \sum_{j=L_{1i}}^{h_{1i}} \sum_{k=L_{2i}}^{h_{2i}} x_{jk} \times M[0, k]. \quad (2)$$

Where:

L_{1i}, h_{1i} are the bounding indices of EBPs in the i th service.

L_{1i}, h_{1i} and L_{2i}, h_{2i} represent the bounds of i th service in the CRUD matrix.

Regarding formula (1) and (2), the average of structural complexity of a service in service set (CRUD matrix), is defined as follows:

$$\overline{SC} = \frac{\sum_{i=1}^S \sum_{j=L_{1i}}^{h_{1i}} \sum_{k=L_{2i}}^{h_{2i}} x_{jk} \times M[0, k]}{S} \quad (3)$$

Where:

S : Number of services.

IV. SERVICE AUTONOMY

Autonomy has various names and definitions. It has been interpreted as self-contained, self-controlling and self-governing [16]. Service autonomy emphasizes logics governed by a service which has an explicit reside boundary. This definition, therefore, controls service in this boundary and it is not dependent on other services for its governance. Service autonomy is one of the primary considerations when deciding on how application logic should be divided up into services and which operations should be grouped together within a service. Service autonomy has a great impact on other attributes of a service, like Discoverability, Reusability and statelessness [17]. This is shown in Figure 1:

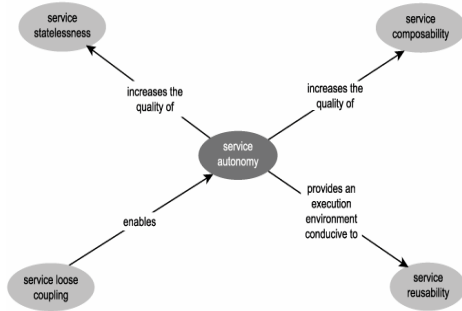


Figure 1. Service autonomy and its relationship with other service-orientation principles [17].

In this section, by considering the importance of Autonomy we will quantify this service quality attribute. According to the definition of a service in previous section, service autonomy is defined as follows:

A service is autonomous if it does all create, read, update, and delete actions related to its own internal BEs without need for communicating with other services. To calculate autonomy, first a coefficient between (0, 1) is assigned to each of C, R, U and D action proportional to their priorities (C>U>R>D) [15]. Then, two metrics for calculating the Autonomy of service are provided.

Metric1: the amount of being self-contained (SLC).

SLC represents the amount of control a service has over the operations related to service's BEs. SLC is defined as follows:

$$SLC = \frac{1}{h_2 - l_2 + 1} \sum_{i=l_2}^{h_2} \sum_{sr \in SR} (BE_{i,sr} \times V_{sr}) \quad (4)$$

where:

V_{sr} : the value of the action sr 's coefficient, as determined by Table II.

$BE_{i,sr} = 1$: If there is at least one EBP in service which does the sr action on i th business entity.

TABLE II. CRUD ACTIONS AND THEIR CORRESPONDING COEFFICIENT

Action	Coefficient
Create(C)	0.4
Read(R)	0.1
Update(U)	0.3
Delete(D)	0.2

Metric 2: Dependency (DEP). As stated before, low coupling improves service autonomy. Coupling represents the

dependency of each service on other services in order to perform its operations. In accordance to this definition, we define dependency metric as follows:

$$DEP = \frac{\sum_{j=L1_i}^{h1_i} \sum_{k=1}^{\#BE} V_{sr,jk} - \sum_{j=L1_i}^{h1_i} \sum_{k=L2_i}^{h2_i} V_{sr,jk}}{nc} \quad (5)$$

where:

$V_{sr,jk}$: is the corresponding value of the action in jk th element of CRUD matrix from table II.

nc : Total number of i th service relations to other services.

$L1_i, h1_i$ are the bounding indices of EBPs in the i th service.

$L1_i, h1_i$ and $L2_i, h2_i$: bounds of the i th service.

The result of calculation of this formula represents the level of dependency between the service i and other services.

Having defined SLC and DEP the autonomy of service is calculated as follow:

$$TSA = \begin{cases} SLC - DEP & SLC \geq DEP \\ 0 & Otherwise \end{cases} \quad (6)$$

Where:

SLC: is the amount of service ability to perform related functionalities.

DEP: The amount of i th service communication (relation) with other services to perform its own functionalities.

V. EXAMPLE

This section describes the presented metrics in details, using an example. Business scenario of the case study is described in [22]. CRUD matrix of a Sale Scenario, described in [22], is provided Figure 2. Also identified services are shown as clusters on this matrix with different colors.

BE \ EBP	customer	Credit	Account receivable note	Order	Discounts	Invoice	Shipping schedule	Draft	Inventory	Warehouse voucher
BE Complexity	21	25	18	12	17	32	20	20	14	23
Add Customer	C	C								
Add an Account receivable note	R	U	C			R				
Check Credit	R	R								
Receive order	R			C						
Calculate discounts				R	R					
Check inventory				R					R	
Calculate price				R	R					
Add discounts					C					
Issue invoice	R	R		R		C				
Schedule shipping						R	C			
Issue draft						R	R	C		
Add an Item									C	
Add a warehouse voucher	R					R			U	C

Figure 2. Sale scenario CRUD matrix [14]

Next, we will describe how to calculate metrics of structural complexity and Autonomy. In order to calculate structural complexity, we determine complexity of BEs according to section III. For example, complexity of Customer and Order BEs are shown in Table III, IV respectively. Complexity of other BEs are provided in row 0 (BE complexity) of CRUD matrix.

TABLE III. COMPLEXITY OF CUSTOMER BE

Customer		
Attribute	Data Type	Structural complexity
Customer ID	Integer	1
Name	String	5
Last name	String	5
Address	String	5
Phone No.	String	5

TABLE IV. COMPLEXITY OF ORDRE BE

Order		
Attribute	Data Type	Structural complexity
Order ID	Integer	1
Status	String	5
Customer ID	Integer	1
Date	Date	5

According to the formula (1), (2), (4) and (5), the method of calculating structural complexity and autonomy of service 1 are described in Table V, VI respectively.

TABLE V. CALCULATING THE STRUCTURAL COMPLEXITY OF SERVICE 1

Service 1(Structural complexity)	
EBP1 Structural complexity	$(BE1\text{Structural complexity} * 1) + (BE2\text{Structural complexity} * 1) = 21 * 1 + 25 * 1 = 46$
EBP2 Structural complexity	$(BE1\text{Structural complexity} * 1) + (BE2\text{Structural complexity} * 2) + (BE3\text{Structural complexity} * 1) = 21 * 1 + 25 * 2 + 18 * 1 = 89$
EBP3 Structural complexity	$(BE1\text{Structural complexity} * 1) + (BE2\text{Structural complexity} * 1) = 21 * 1 + 25 * 1 = 46$
Service Structural complexity	$46 + 89 + 46 = 181$

TABLE VI. CALCULATING THE AUTONOMY OF SERVICE 1

Service 1(Autonomy)		
SLC	SLC with regard to BE_1	$1 * 0.4 + 1 * 0.1 = 0.5$
	SLC with regard to BE_2	$1 * 0.4 + 1 * 0.3 + 1 * 0.1 = 0.8$
	SLC with regard to BE_3	$1 * 0.4 = 0.4$
	Total SLC of service 1	$SLC = (0.5 + 0.8 + 0.4) / 3 = 1.7 / 3 = 0.56$
DEP	$DEP = \frac{(1.7 + 0.1 + 0.1) - (1.7)}{2} = 0.1$	
TSA	$TSA = SLC - DEP = 0.56 - 0.1 = 0.46$	

Calculation results for all services are provided in Table VII.

TABLE VII. RESULT OF ALL CALCULATION

Services	Structural complexity	Structural complexity in Scale(0-1)	Autonomous
Service 1	181	$181 / 501 = 0.36$	0.46
Service 2	99	$99 / 501 = 0.19$	0.40
Service 3	156	$156 / 501 = 0.31$	0.36
Service 4	65	$65 / 501 = 0.12$	0.45
Average structural complexity of a service in service set (CRUD matrix)		$501 / 4 = 125.25$	
Average Autonomy of a service in service set (CRUD matrix)		$1.67 / 4 = 0.41$	

VI. EVALUATION OF PROPOSED METRICS

A. Theoretical Evaluation of Proposed Structural complexity Metric using Weyuker's Properties

Weyuker [18] presented a framework to evaluate structure complexity which is in a form of attributes. Proposed metric in this paper for structure complexity is evaluated using the following attributes. Based on the approach of [19] the properties are:

Property 1: There are programs P and Q for which $|p| \neq |Q|$.

Property 2: If c is non-negative number, then there are finitely many programs P for which $|p| = c$.

Property 3: There are distinct programs P and Q for which $|p| = |Q|$.

Property 4: There are functionally equivalent programs P and Q but $|p| \neq |Q|$.

Property 5: For any program bodies P and Q , we have $|p| \leq |p; Q|$ and $|Q| \leq |p; Q|$.

Property 6: There exist program bodies P , Q and R such that $|p| = |Q|$ and $|p; R| \neq |Q; R|$.

Property 7: There are program bodies P and Q such that Q is formed by permuting the order of statements of P and $|p| \neq |Q|$.

Property 8: If P is a renaming of Q , then $|p| = |Q|$.

Property 9: There exist program bodies P and Q such that $|p| + |Q| < |p; Q|$.

These properties are evaluated for the proposed metric as:

- There may be two different Services with different complexities, thus satisfying property 1.
- As Service will have at least one EBP with some functionality, its structural complexity will always have some positive value, thus satisfying property 2.
- For two different Services with different functionalities, structural complexity value may be the same, as these EBP may have the same interface structure but with different functionalities. Property 3 is satisfied on the proposed structural complexity metrics.
- For two different services with the same functionalities there may be BEs with different attributes in service which affect service structural complexity and results in difference between two services, thus satisfying property 4.
- The structural complexity of any composite service S formed by services p and q is more than or equal to the structural complexity of each of them, since S contains the EBPs in both p and q , thus satisfying property 5.
- For two services with the same structural complexity if service R combines with services P and Q there may exist some EBPs of service R in P or Q , so the structural complexity of composition of R and P might differ from the composition of R and Q , thus satisfying property 6.
- The order of EBPs in a service will not change the structural complexity of the new Service, thus property 7 is not satisfied by our structural complexity metrics.

- It is obvious that renaming an EBP or a Service will not affect the structural complexity of that Service, thus satisfying property 8.
- By composing two services, P and Q , if there is a relation between them, structural complexity of this relation will be added to structural complexity of the composed service but it will not have any impact when P and Q are individual services, thus satisfying property 9.

B. Empirical Validation

To empirically validate our metric, we used the analysis of a group of 10 experts in the SOA field on 10 different services. The group was consisted of the experts registered for a complementary graduate degree in Shahid Beheshti University.

There are two main approaches for data analysis, quantitative and qualitative [20]. In this scenario quantitative analysis was used. The subjects were asked to assign a number between 0 and 10 as the cohesion of a service. Since the experiment was distribution free, Spearman Rank-Difference Correlation Coefficient [21], r_s , was used to calculate the correlation coefficient of the data. Spearman r_s is a non-parametric statistics that is used to show the relationship of two variables which are expressed as ranks [20].

Using Spearman Correlation Coefficient we can show that the Cohesion metric is correlated to each subject's provided ratings individually. Our null hypothesis is:

H_0 : "there is no correlation between the Structural complexity and Autonomy Metric with the subject's rating"

The probability of the null hypothesis to be rejected is controlled by one confidence levels: $\alpha_1 = 0.05$. The decision rule used for rejecting the null hypothesis is:

α_1 : reject H_0 if $r_s \geq 0.648$.

Table VIII shows summary statistics describing the Spearman rank-difference correlation coefficient between subjects' ratings and the values given by the structural complexity metrics. For each subject, the correlation coefficient, r_s , is given.

TABLE VIII. SPEARMAN RANK CORRELATIONS FOR STRUCTURAL COMPLEXITY

Subjects	r_s	α_1
1	0.803	Reject H_0
2	0.725	Reject H_0
3	0.686	Reject H_0
4	0.336	Accept H_0
5	0.649	Reject H_0
6	0.772	Reject H_0
7	0.635	Reject H_0
8	0.854	Reject H_0
9	0.910	Reject H_0
10	0.761	Reject H_0

Based on Table VIII data, the null hypothesis for 90% of the subjects are rejected, with regards to α_1 . Also, the confidence level for α_1 is 95%. After further analysis it is conquered that there's a strong correlation between the proposed metric and the subject's rating.

Table IX shows summary statistics describing the Spearman rank-difference correlation coefficient between

subjects' ratings and the values given by the autonomy metrics. For each subject, the correlation coefficient, r_s , is given.

TABLE IX. SPEARMAN RANK CORRELATIONS FOR AUTONOMY

Subjects	r_s	α_1
1	0.732	Reject H_0
2	0.674	Reject H_0
3	0.732	Reject H_0
4	0.828	Reject H_0
5	0.741	Reject H_0
6	0.410	Accept H_0
7	0.779	Reject H_0
8	0.728	Reject H_0
9	0.822	Reject H_0
10	0.773	Reject H_0

VII. CONCLUSION

In this paper, some metrics were presented to measure Structural Complexity and Service Autonomy in service-oriented modeling phase. The main applications of proposed metrics are in metric-based service identification approaches and, evaluation of identified services. It could be concluded that by measuring Structural Complexity and Service Autonomy in modeling phase and providing the right level of these attributes implementation cost could be decreased, due to Reusability and Maintainability enhancement.

Future Researches include improvement over the proposed metrics using feedbacks provided in service implementation phase, and also determining the impact and relationship between these attributes, structural complexity and autonomy, and other quality attributes.

ACKNOWLEDGMENT

The project has been partially supported by Iran Education and Research Institute for Information and Communication Technology (formerly Iran Telecommunication Research Center (ITRC)) under contract number 4290/500 and also Shahid Beheshti University under the supervision Automated Software Engineering Research (ASER) Group, Faculty of Electrical and Computer Engineering. Authors would like to thank Mr. Amin Zamiri, and all the members of Automated Software Engineering Group for their help on reviewing this work and their great support.

REFERENCES

- [1] Q.Zhang, and X.Li, "Structural complexity Metrics for Service-Oriented Systems". *Second International Symposium on Knowledge Acquisition and Modeling*, 7-10. doi: 10.1109/KAM.2009.
- [2] N. Salman, "Structural complexity Metrics AS Predictors of Maintainability and Integrability of Software components". *Journal of Arts and Sciences*, 2006, pp. 39-50.
- [3] J. Cardoso, "Control-flow Structural complexity Measurement of Processes and Weyuker's Properties," *Engineering and Technology*, 2005, pp. 213-218.
- [4] V. Gruhn and R. Laue, "Structural complexity metrics for business process models," *In 9th International Conference on Business Information Systems (BIS)*, 2006.
- [5] S. Mahmood and R. Lai, "A structural complexity measure for UML component-based system," *Software-Practice and Experience*, 2008, pp. 117-134.

- [6] B. Shim, S. Choue, S. Kim, and S. Park, "A Design Quality Model for Service-Oriented Architecture *," *15th Asia-Pacific Software Engineering Conference*, 2008, pp. 403-410.
- [7] Q.P. Thi, D.T. Quang, and T.H. Quyet, "A Structural complexity Measure for Web Service," *International Conference on Knowledge and Systems Engineering*, IEEE, 2009, pp. 226-231.
- [8] M. Hirzalla, J. Cleland-huang, and A. Arsanjani, "A Metrics Suite for Evaluating Flexibility and Structural complexity in Service Oriented Architectures," *Service-Oriented Computing-ICSOC*, 2008.
- [9] A. Arsanjani, "Service-Oriented Modeling and Architecture (SOMA)", IBM® developer Works, 2004.
- [10] P. Jamshidi, S. Khoshnevis, R. Teimourzadegan, A. Nikraves, and F. Shams, "Toward Automatic Transformation of Enterprise Business Model to Service Model," *Vancouver, Canada: ICSE'09 Workshop*, 2009, pp. 70-74.
- [11] F. Shams, P. Jamshidi, S. Khoshnevis, A. Nikraves, R. Teimourzadegan, A. Khoshkbarforousha, "A Framework for Automatic Transformation of Enterprise Business Model to Service Model", *Automated Software Engineering Research Group*. <http://aser.sbu.ac.ir>, 2009.
- [12] M. Oara, "Finding Services in the Mainframe," *SOA Magazine*, vol. 2007, pp. 1-7.
- [13] P. Jamshidi, M. Sharifi, S. Mansour, "To Establish Enterprise Service Model from Enterprise Business Model", *International Conference on Services Computing*, IEEE, 2008.
- [14] S. Khoshnevis, P. Jamshidi, A. Nikraves, A. Khoshkbarforousha, R. Teimourzadegan, F. Shams, "ASMEM: A Method for Automating Model Evolution of Service-Oriented Systems", *3rd International Workshop on a Research Agenda for Maintenance and Evolution of Service-Oriented Systems (MESOA)*, 2009.
- [15] P. Jamshidi, S. Khoshnevis, R. Teimourzadegan, A. Nikraves, and A. Khoshkbarforousha, "ASSM : Toward an Automated Method for Service Specification," *IEEE*, 2009, pp. 451-456.
- [16] Y. Ma, H.X. Li, and P. Sun, "A Lightweight Agent Fabric for Service Autonomy," *Springer*, 2007, pp. 63-77.
- [17] T. Erl, *Service-Oriented Architecture Concepts, Technology, and Design*, Prentice Hall PTR, 2005.
- [18] E. Weyuker, "Evaluating software structural complexity measures," *IEEE Transactions on Software Engineering*, vol. 14, 1988, pp. 1357-1365.
- [19] A. Sharma, R. Kumar, and P.S. Grover, "Empirical Evaluation and Critical Review of Structural complexity Metrics for Software Components," *6th WSEAS Int. Conf. on Software Engineering, Parallel and Distributed Systems*, 2007, pp. 24-29.
- [20] J. Cardoso, "Process control-flow structural complexity metric : An empirical validation," *IEEE International Conference on Services Computing (IEEE SCC 06)*, Chicago: IEEE, 2006, pp. 167-173.
- [21] S. Siegel, and J. N. John Castellan, *Nonparametric Statistics for The Behavioral Sciences*, McGraw-Hill, 1988.
- [22] P. Jamshidi, S. Mansour, "ASIM: Automated Service Identification Method", *IEEE Transactions on Service Computing* (under review)
- [23] A. Rostampour, A. Kazemi, F. Shams, A. Zamiri, P. Jamshidi, "A Metric for Measuring the Degree of Entity-Centric Service Cohesion" *International Conference on ServiceOriented Computing and Application(SOCA)* " ,IEEE, 2010.
- [24] A. Kazemi, A. Rostampour, F. Shams, P. Jamshidi, and A. Nasirzadeh Azizkandi, "Measuring Service Cohesion Using Latent Semantic Indexing.", *The Sixth International Conference on Internet and Web Applications and Services (ICIW)*, 2011.(Accepted)