Failure Recovery of Composite Semantic Web Services by Subgraph Replacement Strategy

Hadi Saboohi, Amineh Amini and Hassan Abolhassani

***Abstract*-- Web services foster functionality of current web to service oriented architecture. Additionally, the semantic web service architecture supports better service invocation by agents because the underlying ontologies are extensible. A middle agent (broker) simplifies the interaction of service providers and service requester, especially in the case that an atomic web service cannot fulfill user's need. It composes a desired value-added service and orchestrates the execution of bundled sub-processes. It is inevitable that several constitutive web services may fail during the execution and become unavailable. In this paper, we propose replacement of a sequence of semantic web services in lieu of old composition subgraph which includes perished web service(s). We try to perform finding foreseeable replacing graphs, and their compatible alternative subgraphs and ranking of them before exploitation of composite web service. Furthermore, we illuminate an approach for resolving functional differences between old and new subgraphs.**

***Index Terms*— Composite Semantic Web Service, Failure Recovery, Web Service Subgraph Replacement**

I. Introduction

SERVICE Oriented Architecture is an architectural paradigm for components of a system and interactions or patterns between them. In this architecture, a service is a contractually defined behavior that can be implemented and provided by a component for use by another component.

Services are described by descriptors. The service description consists of the technical parameters, constraints and policies that define the terms to invoke the service. Each service should include a service definition in a standardized format. This enables applications and human actors to examine the service description and determine issues such as, what the service does, how they may bind to it, and what security protocols (if any) must be used with it. A service must communicate its service description in an accessible manner to potential consumers. It does so by using one of several advertising methodologies [1].

The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation [2]. When looking towards the future of web services, it is predicted that a breakthrough will come when the software agents start using the web services rather than the users who need to browse, discover and compose the services. Providing the semantic of web services give the software agents the capability to discover and compose web services. The Semantic Web enables greater access not only to content of the Web but also to services on the Web [3].

In a service oriented environment, to achieve user's goals it is needed to find appropriate semantic web service(s). Furthermore, if Service Registry does not include desired web service, Composer component, like one defined in [4], composes existing web services and exploits it as a composite web service to satisfy user's goal. A composite web service is a combination of smaller services to provide value-added services that cannot be achieved by a single service. Of which some of this smaller service can be composite services as well.

Executing of a composite semantic web service includes execution of all bundled services. So, a composite service is more suspectable to failure than an atomic service.

During the execution of a composite web service, if one component service fails, or becomes unavailable, a mechanism is needed to ensure that the running process is not interrupted and the failed service is quickly and efficiently replaced. Furthermore, considering transactions, if composite web service fails at a point, well-executed services of this structure must undo [5] and roll-back to starting state.

When a composite web service fails, we may try to complete its execution by attempting to re-execute failed constituent web service(s) as the first solution. If re-execution attempts are unsuccessful, we apply a failure recovery strategy by replacing a sequence of web services containing this jammed service(s) to complete the execution.

This paper is organized as follows: The next section summarizes related works on this topic. Section III outlines our method for replacing a sequence of web services and their replacement patterns. Two instance scenarios in section IV shows the functionality of the method. Finally, we provide conclusions and ideas for future research.

II. Related Works

Mechanisms are being developed to recover from failure automatically. One approach for recovering from faults is sug-

Hadi Saboohi and Amineh Amini are with the Department of Computer Engineering, Karaj Islamic Azad University, Karaj, Tehran, Iran (e-mail: saboohi@kiau.ac.ir and aamini@kiau.ac.ir).

Hassan Abolhassani is with the Department of Computer Engineering, Sharif University of Technology, Tehran, Iran (e-mail: abolhassani@sharif.edu).

gested in [6]. It proposes a novel taxonomy that captures the possible failures that can arise in web service composition and classifies the faults that may cause them. The taxonomy covers some faults that can be caused by a variety of observable failures in a system's normal operation. An important usage of taxonomy is identifying the faults that can be excluded when a failure occurs.

A team-oriented model introduced in [7] has the ability to coordinate autonomous components in face of a failure. That research examined how team-work can be used to provide both forward and backward error recovery as well as efficient failure recovery in composite web services.

From a different approach, an exception resolving method based on discovering replacement components that are functionally equivalent, taking also into account criteria for qualitative substitutability is proposed in [8]. This solution also introduces a framework called Service Relevance and Replacement Framework which undertakes exception handling.

In [9], authors presented two algorithms to solve the problem of failing or overloading of a component during the execution of an autonomic process. The first algorithm uses a back-up path approach so that the predecessor of a failed service may quickly switch to a predefined backup path. The second algorithm uses a replacement path approach to re-construct a new process by skipping the failed service.

A solution offered in [10], employ a pre-processor that enhances Business Process Execution Language scenarios with code for intercepting faults and invoke alternate web services. Identification of same skilled alternatives is based on both functional and qualitative attributes.

In this paper, we are not interested in monitoring and detecting failures like most researches do. Instead we are more interested in how to recover from a composite web service failure. Related works mentioned above replace or skip a single web service when a failure happens. On the other hand, we suggest replacing a sequence of web services to recover the failure.

III. Failure recovery method

A. Method Description

In a service oriented architecture, processes are done by invocation and execution of web services. Composing atomic (and other composed) web services is used when an appropriate web service to satisfy user's goal is not exists. Web service composition does not involve the physical integration of all components: The basic components that participate in the composition remain separated from the composite web service.

Fig. 1 [4] suggests a service brokering architecture. As shown in the figure, a service broker places between service providers and requester as a middle agent. Service requester sends its request to the broker. Matchmaker searches Service Registry using Ontology Manager and Ontology Cache to find nearest web service regarding degree of similarity to the request. If no similar web service found, Composer tries to compose a new value-added service and sends the result to Matchmaker. Composite web services information caches in Composed Services Cache. Executor interacts with Service Providers and sends required information for them and receives the result(s). Adapter adapts results with user's need and sends them to Service Requester [4].

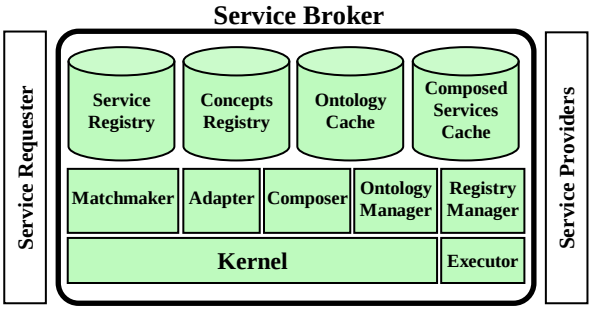

Fig. 1. Service Broker [4]

The broker orchestrates the execution of bundled sub-processes. However, as business processes are typically long-lasting transactions, it is inevitable that several constitutive web services may fail during the execution and become unavailable. So, all aspects of executions should be considered to achieve user's goal and rectify the problem situation. Executor component of the broker undertakes inability to succeed and struggles to hinder failure of software system. Complete execution of composite web service need execution of all constitutive web services. In database system concepts, transactions are used to ensure that integrity and consistency of databases.

Transactionality, in particular the atomicity, consistency, isolation, and durability (ACID) represent a prevalent approach for solving reliability issues in distributed computing. ACID transactions are usually implemented using transaction monitors or component platforms. Support for ACID transactions requires coupling through the transactional environment, thus limiting interoperability and flexibility. Another requirement for ACID transactions implementation is resource locking for the duration of the transaction, which requires guaranteed short execution time of services. Longer transaction time usually leads to worsening of the overall throughput of transactional resources. ACID transactions, while perfectly appropriate for objects and components, are usually too restrictive for services [11]. Components of composed web service may use transactions internally.

The notion of compensating transactions offers a way to undo an action if a process or user cancels it. With compensating transactions, each method exposes an undo operation that a transaction coordinator can invoke if necessary [12].

In general, a composite web service can be modeled as a directed graph in which web services are nodes and their inputs and outputs are edges. Our approach in modeling and mapping composite web services to a directed acyclic graph is similar to [9]. However, in our research we use a simple sequential graph which is defined below.

Definition 1: *A S-Graph (Sequential Graph) is a directed, planar, acyclic, linear graph with one start and one end nodes. Each node is a web service and has input degree and output degree of one except start and end nodes. The order (number of nodes) of graph is one bigger than its size (number of edges).* $|G| = \|G\| + 1$ *or* $|V(G)| = |E(G)| + 1$

Execution of each web service depends on execution of only one other web service, which means web services are executed one after the other.

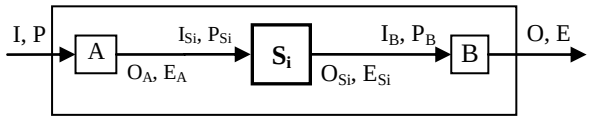

Fig. 2. Composite Web Service (A sample S-Graph)

A composite web service structure is as shown in Fig. 2.

Problem Definition: In a composite semantic web service structure, execution of whole process depends on well-execution of all sub-processes. Executor component of used architecture (Fig. 1) orchestrates the execution task. Web services as other software system components are error-prone. So, the Executor should react if a failure occurred in execution of any of the constitutive web services(s). Failure recovery is a better solution in compare with halting the software system. One possible approach to conquer the failure is to replace a sequence of web services containing failed service. Any replacement strategy in this area is responsible for the followings:

- Finding best substitution alternative among candidates (if many).
- Managing differences between old and new service(s).

In our method, if a failure occurred during execution of a composite web service, we replace the failed web service with a "same skilled" [8] one, and if there is no "same skilled" web service, a subgraph of web services including this failed one can be replaced to continue execution of composite web service.

The process of finding replacement includes two steps, which both executed offline so replacement speed improves.

1) Step 1: Subgraph Calculation and Alternative Search

We assume that the structure of composite web service is available in Composed Services Cache (Fig. 1). From that, first, we calculate all possible subgraphs of composite web service. Number of subgraphs for those as in Definition 1 is:

$$Sub\,graphs\,Count = \frac{n(n+1)}{2} \quad (1)$$

For example, a set of subgraphs of the sample S-Graph in Fig. 2 are as: {A}, {S_i}, {B}, {A, S_i}, {S_i, B}, {A, S_i, B}

Then, we search for replacement alternatives of these subgraphs which are compatible to the original one. (Here, by compatibility we mean, the alternative has both functional and non-functional attributes so it is compatible by one of Replacement Patterns of next section). It should be mentioned that discovered replacement alternatives may have different number of constructive services than the original subgraph.

Achieving the desired outcome at an accelerated pace in the following step 2, we utilize an indexing method for list of web services in current step.

2) Step 2: Ranking Alternatives

In step 1's result table, each web service of composite web service may exist in more than one subgraph. For example $\mathbf{S_i}$ in figure 2, is in four subsets of subgraphs ({S_i}, {A,S_i}, {S_i,B}, {A,S_i,B}). Furthermore, for each subgraph, several alternatives may exist. So, it is probable to have many replacement subgraph alternatives for a subgraph containing $\mathbf{S_i}$ web service.

In this step, all replacement subgraphs of a web service are grouped and ranked by a linear combination of five criteria as follows:

$$Rank = \alpha_1 Undo_{Count} + \alpha_2 Undo_{Cost} + \beta_1 New_{Length} + \beta_2 New_{Cost} + \delta Differences \quad (2)$$

This measure depends on number of web services need to be compensated in the original graph ($Undo_{Count}$), Undo cost of the original subgraph ($Undo_{Cost}$), Number of web services in new subgraph (New_{Length}), Execution cost of replacing new subgraph (New_{Cost}) and functional and non-functional *Differences* between the original and replacing new subgraph. These five criteria are combined with different weights specified by α_1, α_2, β_1, β_2 and δ. These weight values are experimentally determined.

In each group, best ranked alternative subgraph, would be the replacement selection if the web service of that group fails.

3) Replacement

During execution, if a web service like $\mathbf{S_i}$ in Fig. 2 fails, we execute it repeatedly until successful execution or re-execution attempts reaches maximum retry count or valid time exceeds. Maximum retry count and time are specified by web service provider. If execution doesn't succeed, we replace a subgraph of web services based on step 2. Furthermore, total time for failed web service re-execution, replacement of new subgraph, running of it and other remaining web services should not exceed the specified valid execution time of composite web service.

Topmost ranked replacement option of the failed web service is easy to reach, using result table of step 2. In replacement subgraph, well-executed web services before the failed web service should be compensated. So, we undo them, and then we send required requests (if any) which are specified in

TABLE I
REPLACEMENT PATTERNS

failed and *new* Differences				Required Task before Execution of *new* Web Services and After Compensation of executed ones in *failed*	Mark new Composite Web Service as non-optimized	Is *new* Appropriate?
Inputs	Preconditions	Outputs	Effects			
–	–	–	–	None	–	Yes
–	$\mathbf{P}_{new} \subseteq \mathbf{P}_{failed}$	–	–	None	❋	Yes
–	$\mathbf{P}_{failed} \subseteq \mathbf{P}_{new}$	–	–	Prepare new preconditions	–	Maybe
–	–	–	$\mathbf{E}_{new} \subseteq \mathbf{E}_{failed}$	Inspect if it is executable without old effects	–	Maybe
–	–	–	$\mathbf{E}_{failed} \subseteq \mathbf{E}_{new}$	None	❋	Yes
$\mathbf{I}_{new} \subseteq \mathbf{I}_{failed}$	–	–	–	None	❋	Yes
$\mathbf{I}_{failed} \subseteq \mathbf{I}_{new}$	–	–	–	Request new inputs	–	Maybe
–	–	$\mathbf{O}_{new} \subseteq \mathbf{O}_{failed}$	–	Inspect if it is executable without old outputs	–	Maybe
–	–	$\mathbf{O}_{failed} \subseteq \mathbf{O}_{new}$	–	None	❋	Yes

the following Replacement Patterns section to provide new requirements. After that, old subgraph is substituted with the new one. Finally, web services in new subgraph are executed to continue composite web service execution.

B. Replacement Patterns

We proposed subgraph replacement for failure recovery of a composite web service. The subgraph which will be replaced with a new subgraph may have functional and non-functional differences. Non-functional or qualitative differences include response time, availability, reliability, cost, encryption, reputation and authentication. Functional attributes of web services are Inputs (**I**), Preconditions (**P**), Outputs (**O**) and Effects (**E**). Different kinds of functional differences between new subgraph (*new*), and old subgraph containing failed web service (*failed*) are shown in table I.

We consider qualitative attribute differences such as execution cost between *new* and *failed* in ranking alternatives.

If *failed* and *new* have same functional attributes, for example another provider created a similar web service(s) or composer generated an analogous composite one, replacement has no side effect on overall structure.

As shown in table I, if there is a difference in one of functional attributes, in almost half of the cases, replacement task is feasible, while in others replacement possibility depends on response value of indicated requirement requests.

Additionally, sometimes we mark the new composite structure as a non-optimized composite web service due to creation of unused output(s) or effect(s), or unnecessary input(s) or precondition(s). This sign which has done in Composed Services Cache component (of used architecture), will be used for future need for this composite web service. This indication is also applicable for reliability checking of the old composite web service.

Differences in more than one of functional attributes explored above are imaginable. It means that *failed* and *new* can have several differences in their attributes. If *failed* and *new* have multiple differences, we will do all (union aggregation of) required tasks, and after receiving responses taking into account minimum changes to composite web service, we can decide the *new* is appropriate or not. Furthermore, if at least one of multiple differences show that old composite web service is non-optimized, we (mark it and) no longer use it as an optimized one.

IV. INSTANCE SCENARIOS

These example scenarios are provided to clarify the function of above method.

A. Conference Registration

Imagine you need a semantic web service to register an international foreign conference and you want to attend there. Contacted Service Broker, after searching the information of registered services in Service Registry, finds out that it doesn't have any such services. So it asks its Composer component to check whether it is possible to integrate existing services for creating a new service which can respond to the request. The Composer component detects that by integrating some atomic services it can do the required job. The execution plan sent to the Executor component by the Composer is shown in Fig. 3.

Composite semantic web service consists of
1. Conference registration fees payment (P)
2. Registration in conference (R)
3. Get a Visa (V)
4. Two-way airline ticket buying (T)
5. Hotel reservation (H)

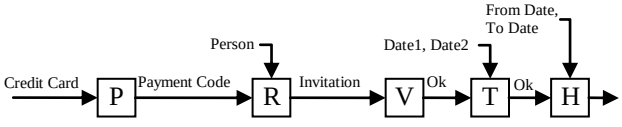

Fig. 3. Conference Registration Composite Semantic Web Service

Total execution cost of this composite web service is 9 units. The first three ones cost 1 unit and the others, 3 units per

each web service.

All subgraphs of PRVTH graph are calculated (shown in table II).

TABLE II
SCENARIO A, PRVTH COMPOSITE SEMANTIC WEB SERVICE SUBGRAPHS

All subgraphs for Instance Scenario PRVTH graph				
{P}	{R}	{V}	{T}	{H}
{PR}	{RV}	{VT}	{TH}	
{PRV}	{RVT}	{VTH}		
{PRVT}	{RVTH}			
{PRVTH}				

Number of subgraphs according to (1) is as below.

$$Sub\,graphs\,Count = \frac{5*6}{2} = 15$$

For each of these subgraphs, possible replacement subgraphs are specified in table III. Some of these replacements are atomic web services and others are composite ones.

TABLE III
REPLACEMENT ALTERNATIVES FOR ORIGINAL SUBGRAPHS OF SCENARIO A

Original Subgraph	Original Subgraph Execution Cost	Replacement Subgraph	New Subgraph Execution Cost
T	3	T_1T_2	4
TH	6	Tour	8
PR	2	R'	3
RV	2	V'	3
PRV	3	A	5

Groups of replacement subgraph alternatives for each feasible failing web service are ranked (Table IV).

During execution time of composite web service PRVTH, if one of its five web services fails, best ranked replacement subgraph is specified in advance. So, using proposed method, it can be replaced.

TABLE IV
REPLACEMENTS RANKING FOR SCENARIO A

Failed Web Service	*failed*	*new*	$Undo_{Count}$	New_{Cost}	Rank
P	PR	R'	0	3	1
P	PRV	A	0	5	2
R	RV	V'	0	3	1
R	PR	R'	1	3	2
R	PRV	A	1	5	3
V	RV	V'	1	3	1
V	PRV	A	2	5	2
T	T	T_1T_2	0	4	1
T	TH	Tour	0	8	2
H	TH	Tour	1	8	1

Table IV is ranked based on $Undo_{Count}$ and New_{Cost}.

B. Physical Therapy Planning

Saba wants to schedule a series of physical therapy sessions for her father's neck pain as his primary care physician suggested. Her handheld agent requests the broker to do the followings: First, retrieve details of the recommended therapy. Next, look up the list of therapists maintained by Health Insurance Company and check which of them are located near her father's house with acceptable reputation. At last, match available appointment time with him.

Service Broker says no atomic web service in Service Registry is able to do this job. So, Composer component composes some atomic and composite services to create a new service which can fulfill user's requirements. Composition structure is shown in Fig. 4.

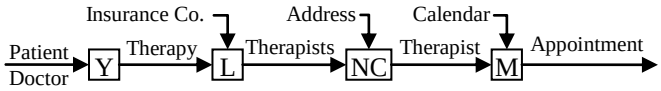

Fig. 4. Physical Therapy Planning Composite Semantic Web Service

These services will be used in that new composite semantic web service:

1. Retrieve recommended therapy service (Y), costs 1 unit
2. Health insurance company therapists list service (L), costs 2 units
3. Neighbor therapists and reputation checker composite service (NC), costs totally 6 units
4. Appointment time matcher service (M), costs 3 units

After calculation of all 10 subgraphs and finding their alternatives, table V are generated.

TABLE V
REPLACEMENT ALTERNATIVES FOR ORIGINAL SUBGRAPHS OF SCENARIO B

Original Subgraph	Original Subgraph Execution Cost	Replacement Subgraph	New Subgraph Execution Cost
Y	1	Y_1	2
Y	1	Y_2	3
NC	6	N_1C	9
LNC	8	Z	12
LNC	8	LN_1C	11

TABLE VI
REPLACEMENTS RANKING FOR SCENARIO B

Failed Web Service	*failed*	*new*	$Undo_{Count}$	New_{Cost}	Rank
Y	Y	Y_1	0	2	1
Y	Y	Y_2	0	3	2
L	LNC	LN_1C	0	11	1
L	LNC	Z	0	12	2
NC	NC	N_1C	0	9	1
NC	LNC	LN_1C	1	11	2
NC	LNC	Z	1	12	3
M	M	-	-	-	?

Table VI shows the rankings of possible alternatives for subgraphs. This table shows that if service M fails, there is no replacement for it. So, by failing M service, executor component must compensate executed services (YLNC sequence from last to first, if no other failures happened before that) and then sends an error message to the requester. In this composite semantic web service structure, NC is a composite one itself. If service C in this smaller composite service fails and Service Registry contains an alternative for C named C_1, there are two options. First, executor can compensate N and execute Z service instead. Second, executor can replace C_1 instead of C and continue the execution process. Failure recovery method described above can be extended through contained services and used recursively in sub-composite services in the structure.

V. Conclusion and Future Works

This paper presented a strategy to alleviate failure of software systems consisting composite semantic web services. It proposed a failure recovery method using subgraph replacement of web services containing a failed web service. This failure recovery method uses both forward and backward mechanisms as followings: First, re-execution of failed web service and second, execution of an alternative subgraph of web services instead of a sequence of services containing failed web service (after compensation of executed ones).

In our method, composite semantic web service is considered to be a simple graph defined as S-Graph but proposed steps are of $O(n^2)$ because the most time-consuming section is the calculation of all subgraphs and finding their compatible alternatives. Future works include using heuristic algorithms to decrease number of subgraphs, and extend this method for all composition graphs of semantic web services. Implementation, calculation of ranking criteria weights and evaluation of the method is underway.

VI. References

[1] D. Nickull. (2005). Service Oriented Architecture white paper, Adobe Systems Incorporated, San Jose, CA, USA. [Online]. Available: http://www.adobe.com/enterprise/pdfs/Services_Oriented_Architecture_from_Adobe.pdf

[2] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web", *Scientific American*, vol. 284, no. 5, pp. 34-43, May 2001.

[3] Dogac, A., M. Eichelberg, L. Finlay, A. Koumpis, B. Kunac, and M. Boniface, "Review of the State of the Art Semantic Web and Web Service Semantics", Software R&D Center, Middle East Tech. Univ., Turkey, Apr. 2004.

[4] Y. Ganjisaffar and H. Abolhassani, "Towards a Framework for Brokering Semantic Web Services" *Int. Journal of Computers and Their Applications (IJCA)*, 2008, To Appear.

[5] M-C. Gaudel. "Toward Undoing in Composite Web Services," In Architecting Dependable Systems III, vol. 3549 of LNCS, pp. 59-68, 2005.

[6] K.S. M. Chan, J. Bishop, J. Steyn, L. Baresi and S. Guinea, "A Fault Taxonomy for Web Service Composition", 3rd Int. Workshop on Eng. Service-Oriented Apps. At 5th Int. Conf. on Service Oriented Computing (ICSOC'07), Springer LNCS, Vienna, Austria, Sep. 2007.

[7] Y. Y. Chok, "Team-oriented Model for Composite Web Services Failure Recovery", Honours Programme of the School of Computer Science and Software Eng., Univ. of Western Australia, 2005.

[8] K. Christos, V. Costas, G. Panayiotis, "Towards Dynamic, Relevance-Driven Exception Resolution in Composite Web Services", 4th Int. Workshop on SOA & Web Services Best Practices, Portland, Oregon, USA at OOPSLA, 2006.

[9] T. Yu and K. J. Lin, "Adaptive algorithms for Finding Replacement Services in Autonomic Distributed Business Processes." in *Proc. 2005 of the 7th Int. Symposium on Autonomous Decentralized Systems (ISADS2005)*, Chengdu, China.

[10] K. Christos, V. Costas and G. Panayiotis, "Enhancing BPEL scenarios with Dynamic Relevance-Based Exception Handling," in *Proc. 2007 IEEE Int. Conf. on Web Services (ICWS)*.

[11] B. Lublinsky. (2007, Jan.). Defining SOA as an architectural style. IBM developerWorks, USA. [Online]. Available: http://www.ibm.com/developerworks/architecture/library/ar-soastyle/

[12] C. Peltz, "Web Services Orchestration and Choreography," *Computer*, vol. 36, no. 10, pp. 46-52, Oct. 2003.

VII. Biographies

Hadi Saboohi was born in Shiraz, Iran on January 25, 1978. He received his M.Sc. on Computer Engineering, Software from the Najafabad Islamic Azad University of Iran with a thesis on Intelligent Content Management Systems. His areas of academic research include Semantic Web, Information Retrieval and Data Integration.

His employment experience included the Iranian Research Organization for Science and Technology and Delaram Pardazeshgar Company developing high-level software systems. He is now a faculty member of Karaj Islamic Azad University, Computer Engineering Department.

Amineh Amini was born in Mashhad, Iran on August 29, 1978. She received her M.Sc. on Computer Engineering, Software from Najafabad Islamic Azad University of Iran with a thesis on Information Integration in Distributed Systems. Her areas of academic research include Semantic Web and Data Integration.

Her employment experience included university lecturer in some of Islamic Azad University and University of Applied Science and Technology branches. She is now a faculty member of Karaj Islamic Azad University, Computer Engineering Department.

Hassan Abolhassani was born in Isfahan, Iran on July 10, 1965. He received his PhD from Saitama University of Japan with a thesis on Automatic Software Design focusing on Learning from Human Designers. His areas of academic research include Software Automation, Semantic Web researches, Web and Data Mining.

He worked as Senior Technologist providing software based solutions for top-level clients in Japan when he was with Xist-Interactive (Razorfish Japan), until the end of September 2004, when he joined Sharif University of Technology as an assistant professor.