SERVICE ORIENTED CONFIGURATION MANAGEMENT OF SOFTWARE ARCHITECTURE

Razie Alidoosti[1], Shahrouz Moaven[2] and Jafar Habibi[3]

[1,2,3]Department of Computer Engineering, Sharif University of Technology

ABSTRACT

Software configuration management (SCM) is an important activity in the software engineering life cycle. SCM by control of the evolution process of products leads to constancy and stability in software systems. Nowadays, use of software configuration management is essential during the process of software development as rules to control and manage the evolution of software systems. SCM effects different levels of abstraction included the architectural level. Configuration of software architecture causes improvement in the configuration of the lower abstraction levels. CM of software architecture is more significant in large scale software with longevity of life cycle. Traditional SCM approaches, at the architectural level, do not provided the necessary support to software configuration management, so systems that use these approaches are faced with problems. These problems arise because of the lack of a serious constant and repeated changes in the software process. To overcome this it is necessary to create an infrastructure. Hence, a service oriented approach for configuration management is presented in this paper. In this approach, the activities of configuration management are conducted from a service oriented viewpoint. This approach was also used to try and control the evolution and number of versions of different software systems in order to identify, organize and control change and reforms during the production process. This approach can compose services and create composite services for new undefined activities of configuration.

KEYWORDS

Software configuration management, Service-oriented approach, Version control

1.INTRODUCTION

The ability to manage the evolution of software architecture is an important issue in the domain of software engineering. In this regard, many methods have been proposed for logical level software systems. These methods present the relationship between the concepts of software configuration management and software architecture [1]. Configuration management for each domain should adopt a comprehensive strategy that includes a set of rules and policies for the specific field.

Configuration management of software architecture demands special considerations. Not only is there revision or change management based on the components, connectors, and their governing rules; but the same discussion is also necessary for architectural artifacts such as software documents, directories, etc. In other words, since there is the possibility of changes being made to

the system at any time it is essential that there be some sort of software configuration management, such as change control and identification, to ensure and report the correct implementation of these changes [2]. In fact, one of the most important objectives of software configuration management is change control which is also one of most effective factors in quality assurance.

A review of activities in the field of software configuration management shows that there have been many attempts in this field. Previous attempts to create models of different configuration management systems and architectures have been based on their infrastructures. The major problem of these infrastructure is mostly due to the lack of mechanisms in different levels of coarse grain [3]. At the architectural level, traditional SCM approaches do not provide the necessary software configuration management support, therefore the systems that use these approaches are faced with problems. These problems arise because of the lack of a serious constant and repeated changes in the software process.

The main goal of this paper is to present a comprehensive model that covers all aspects of configuration management at the architectural level, such as parallel development, distributed engineering, build and release management, change management, and process management. Appling these service-oriented principles in this field can create a comprehensive and efficient infrastructure [4] that could be used for composite services and to create any new services required. In this context, service-oriented principles is important to almost all of the principles, models, and standards used in service-oriented architecture that lead to characteristics, such as loose coupling, abstraction, reuse, heritability, autonomy, etc [5]. Applying these principles may improve some issues in configuration management of the infrastructure such as reusability of fine grain at different levels, scalability, flexibility, availability, reliability, complexity, and efficiency.

The service-oriented approach is facilitated by simplification of the problem, by breaking each problem into smaller problems, analysing the issues, and finally combining the obtained solutions. It can solve many of the problems involved in configuration management [6]. In other words, the service-oriented approach supports granularity at different levels. Basically, service-oriented principles from the service concept supported by standards and technologies, such as web services, can be used to create applications that are quick, cost effective, interoperable, and expansible. It should be noted that the concept of object versioning is also considered together with the service-oriented approach in this model.

The remainder of the paper is structured as follows: Section 2 briefly discusses related concepts about software architecture, configuration management and the service oriented approach. Section 3 summarizes previous related works on software architecture via SCM. Section 4 introduces the service oriented configuration management model. Comparison between previous works and the proposed method have been carried out in Section 5. Section 6 summarizes and concludes the key contributions.

2. BACKGROUND

The architectural model developed in Section 4 relies on concepts from software architecture, CM fields, and the service oriented approach. This following section discusses these concepts.

2.1. Software architecture

With software systems becoming more complex over time, using coarse-grained building blocks for the design has become essential. In this regard, software architecture is used to provide a high level of abstraction to display the structure, behavior and properties of software systems [7]. The architecture provides the infrastructure which is used in the design and implementation process. In fact, architecture is able to control the evolution of the design [8].

2.2. Configuration management

Software configuration management is about managing the evolution of systems [9]. Configuration management is a discipline for recording information about CIs [10]. Indeed, it is a task embedded in the system to control change. Recently, it has been used mainly in systems that are very large and complex [11].

Configuration management is used to maintain the integrity of elements in software projects. Since evolution and change is unavoidable in software systems configuration management is considered as an integral element of software development and activity maintenance [12]. Software configuration management consists of activities such as: change management, version managements, system construction, and release management.

2.3. Service oriented approach

A service-oriented paradigm can be used to enhance non-functional requirements such as flexibility, reusability, extensibility, portability, and maintainability [13]. The service-oriented paradigm can be used in major program management to enhance coordination between different elements and organizing data between various applications. Generally, the final objective of this paradigm is integration of the application's element with lower coupling. In fact, a service oriented approach is a simplifying approach which breaks a problem into sub problems, analyzes them and produces a composition of the results for their solutions. In this approach, the emphasis is on functional decomposition at a coarse level instead of structural decomposition [14].

.Basically, using the service concept from a service-oriented paradigm will support quick, low cost, interactional and extensional application development. Services are a common language for heterogeneous system providing the component that connects them during development so that their information can be shared.

3. RELATED WORK

This section discusses existing SCM-based related works for managing architectural evolution. Given the importance of configuration management as an umbrella activity in the software development cycle, numerous studies have been done in this area.

An environment for architecture evolution, called Mae was introduced in [7]. This environment was created to facilitate an incremental design process and also to combine architectural concepts with concepts from the field of configuration management to be used in a generic system model for integration. Ragnarok [8] presented a software configuration management model which emphasized the traceability and reproducibility of the architecture changes and configurations. One of interesting feature of this model has been implemented in three research prototype in three projects with different sizes. This model used logical software architecture from other cases for a

version and configuration control process. Problems containing the Architectural Run-time Configuration Management (ARCM) system were discussed in [15]. The ARCM system is used to improve the dependability of such systems by developing facilities for adaptation recording, enhancing configuration visibility, and providing user-driven support. The study in [16] explains that the proposed service-oriented accounting configuration management system complies with all requirements defined. The new accounting configuration management system was based on Diameter and is highly suitable for wireless and mobile communication systems.

Molhado [3] introduced a novel approach to manage architectural evolution of software systems at the logical level. It presents a novel approach to managing both planned and unplanned architectural evolution. This architectural system model has extensibility functionality to support different architectural description languages and architectural styles. Existing SCM-based approaches were discussed in [1] to manage architectural evolution and classified those approaches into the following categories: orthogonal integration, SCM-supported software architecture, SCM-centered software architecture, and architecture-centered SCM. Each of these approaches expresses the relationship between architectural concepts and configuration management. Flexible product versioning and a structural configuration management model were introduced in [17] and used in the software concordance environment to manage the evolution of a software project and hypermedia structures by focusing on Molhado's hypertext versioning. This model avoids the complications of version selection rules in composition models and the version proliferation problem in total versioning models. A demonstration in [18] presents an extensible object-oriented approach to managing the evolution of system objects at the logical level. This approach applies in different development paradigms such as UML-based object oriented software development, architecture-based software development, and Web application development.

A novel framework and infrastructure provided in [19] was used to build object-oriented software configuration management services in a SCM-centered integrated development environment. This framework included a product versioning model, object-oriented system model, and a reusable product versioning SCM infrastructure. In other words, it demonstrated the feasibility of an object-oriented approach that could be reused in object-oriented SCM systems. A novel architectural SCM system, called MolhadoArch, is introduced in [20] whose configurations are maintained at all levels of abstraction including the architecture and implementation levels. It presents a system for object-oriented software configuration and version management technology to manage versions of architectural structure/entities, source code, and the traceability relationships among them. An architecture-based approach to runtime software evolution is presented in [21]. The approach highlights the role of software connectors in supporting runtime change. This approach has an important role in mitigating the costs and risks of an update. In [22] the application of that infrastructure used for the construction of a multi-level SCM system for source code and structured documents is described. The study in [23] presented a candidate routing architecture for the Airborne. The primary characteristic of this architecture was to implement many of the networks and platforms as separate AS domains via BGP. This architecture is based on Internet Protocol (IP). In [16] a role model was defined covering all involved entities in a distributed service provisioning environment which provided for mobile networks together with key requirements. A new web services discovery model was proposed in [24] which uses functional and non-functional requirements for the service discovery. This service also discusses issues related to slow take up.

4. SERVICE ORIENTED CONFIGURATION MANAGEMENT MODEL

This section presents a model derived from the service oriented paradigm to improve reusability in different levels of coarse-grained. This model takes advantage of service oriented principles such as loose coupling, abstraction, reusability, composition, autonomous, etc. The use of some of these principles can cause improvement in existing configuration management field issues such as scalability, variability, availability, reliability and complexity. The proposed architectural model shown in Figure 1, which in we have tried to incorporate the principles of service orientation as much as possible. The concept of service can be used in concepts such as component and class, but conceptually is at a higher level of abstraction with respect to objects and components and is not a substitute for them.

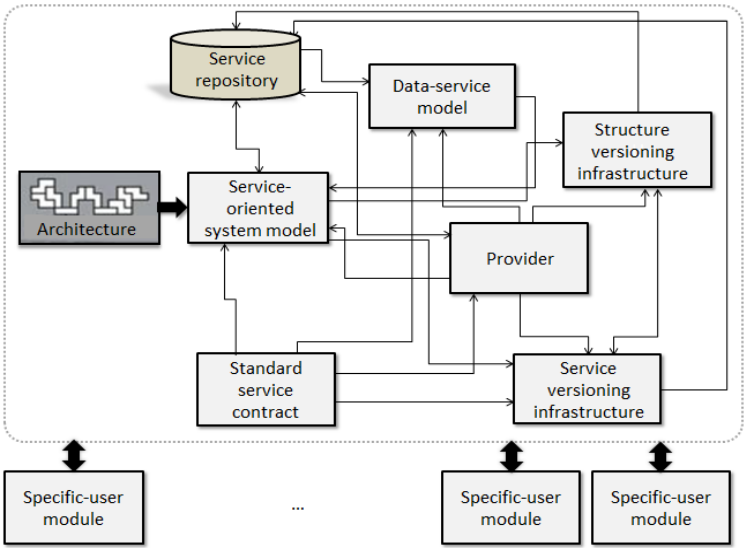


Figure 1. The proposed architectural model

The proposed architectural model includes elements that each plays a special role in the configuration management of architecture. In addition to the service-oriented approach, the concepts of object and versioning are also considered. Elements that are considered in this model include:

A Service-oriented system model; is considered as a framework for modeling a software system. This element receives software system architecture as input. The received architecture decomposes to components and connectors. In the proposed solution for software architecture configuration a system architecture can be mapped to a particular service through different approaches. The first approach considers communication and limitations of the service-oriented model according to the functionality of components in the architecture, and maps one or more components of the architecture together with its connections to the corresponding service. According to the model presented in Figure 1, the most cost effective solution for identifying services, takes advantage of a repository that will be filled over time and used as a source of knowledge for future works.

Services can overlap, be added or even contradict each other. First, we detect services and based on the existing rules in the service-oriented approach, optimize these services. This means that some services are combined or broken down into separated services. Then the well-defined

services are added to the service repository. Some of the services defined for configuration management are shown in Table 1.

Identified services based on architecture layers can be categorized in three groups: process services, basic services and application services. The techniques, principles and modeling languages of the service-oriented paradigm are used in this context. Moreover, this element has the property of extensibility to support the creation of new kinds of software artifacts.
In fact, the element forms an abstract service-oriented model of received system by taking advantage of service repository, data- service model, provider and standard service contract will be discussed later. The formed model is referred to structure versioning infrastructure and service versioning infrastructure in order to save changes and the reconfigurations have been done over several times.

Service repository; services stored in this repository are divided into two categories: 1) basic services identified in the architecture of the systems such as remove service, insert service, overlap service, etc. and 2) the architecture that can be used to configure services such as versioning, change control, identifying CIs, etc. The repository take advantage of the registration and discovery approach for services and also supports various interactive protocols between services. the element is used for retrieving the saved services and maintain the changed services or newly created ones.

Services versioning infrastructures*;* the identifying services may change as the system changes for different reasons, and over time creates new versions of the systems as seen in Figure 2. Some of these changes include contract changes, changes during construction, implement changes, and changes in address. The advantage of loose coupling in the service oriented paradigm can reduce the effect of the variability of a component with other components, and this in turn, can affect the versioning services. It should be noted that this element keeps versions of the existing services. The information related to different versions of services will be sent to repository service.

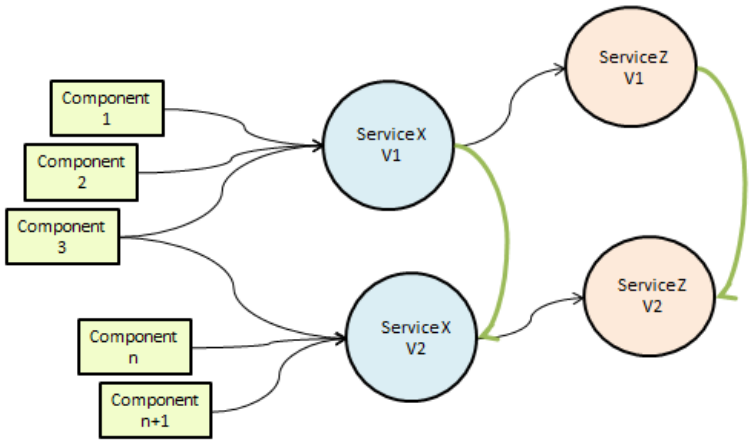


Figure 2. Versioning of services

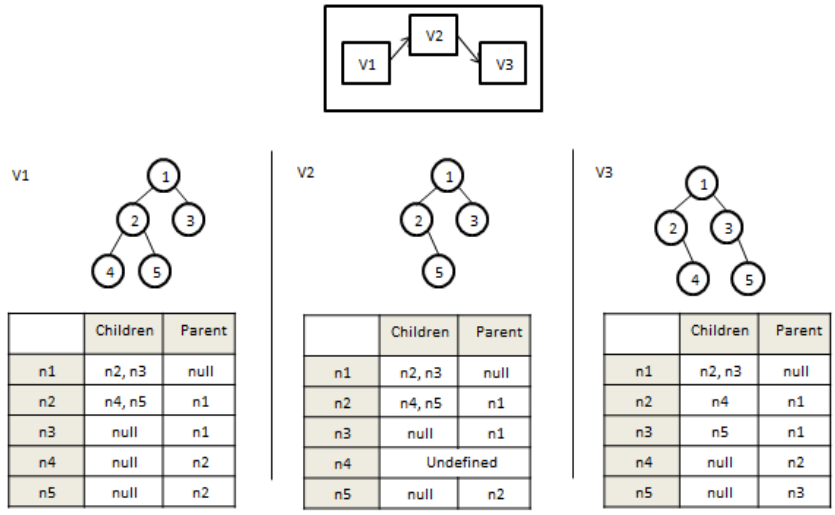
Standard service contract; the relationships between the services are defined according to the contract Properties of this contract are specified in the implementation documents. The considerable points in this contract include purpose of service and functionality capabilities of services. The element declare standards and contracts that is allowed to be used in service-oriented architecture and should be followed in versioning and changes of configuration.

Data-service model*;* this element is used to define new services that can lead to the definition of components and connectors in the architecture model.

User-specific modules; are used by the user to communicate with the mentioned elements. Indeed, they play the role of user interfaces in this architecture.

Structure versioning infrastructure; supports versioning of the created components, connectors and artifacts that includes a series of fine version control algorithms. Figure 3 display the configuration related to an artifact of software architecture. All cases of architecture is versioned and configured based on a series of architectural baselines and aggregated and moved to a higher level, so architecture is versioning and finally, configuring.

We can use this element for each of the mentioned cases in the software system to control and manage different versions. This infrastructure recieves support from software configuration management policies and transactions. The information related to different versions of services will be sent to repository service.



	Children	Parent
n1	n2, n3	null
n2	n4, n5	n1
n3	null	n1
n4	null	n2
n5	null	n2

	Children	Parent
n1	n2, n3	null
n2	n4, n5	n1
n3	null	n1
n4	Undefined	
n5	null	n2

	Children	Parent
n1	n2, n3	null
n2	n4	n1
n3	n5	n1
n4	null	n2
n5	null	n3

Figure 3. Versioning of an artifact of software architecture

Provider; the task of this element is to provide the needed services to perform work. As mentioned above, the identified services are grouped in two categories: 1) services identified based on the functionality of the components and 2) services identified based on the service repository. It also defines new services which utilize and cooperate with the data-service model element. The services are then stored in the repository and the element provides the necessary accesses at required times. The element is used for versioning, configuration and changes has been done in reconfigurable items by using existing services in repository service. In addition, it changes the received architectural model on the basis of configuration services that the changed model will be sent to structure and service versioning infrastructure to versioning.

The dynamic composite of services found in configuration management of software architecture is another activity considered in the provider element and is from main feature of proposed paper. Dynamic composite of services for configuration management of software architecture occurs when we apply reconfigure, which occurs when an event changes in the architectural level. This requires configuration activities to be conducted if there aren't architectural design alternatives.

In these cases several different services together with several application services perform related activities that require a combination of different services. For this purpose, the service composition mechanism is use, this mechanism is described below. The number of services is increased in all areas usually. But it is not possible that service repository and its services respond to all the considered requests. So one way to solve or reduce this problem is to combine existing services to create a new service that is able to satisfy these requirements. In general, the problem of service composition is very difficult. Composition methods of web services are based on the degree of composition of the production process and are divided into three categories: automatic, semi-automatic and manual. We considered an automatic method here. In the automatic methods, the goal is to produce a perfect combination without human intervention. In these methods the requirements are expressed in a specific language, and then produce a combination that tries to meet these requirements. In cases where more than one composition can meet the requirements it is necessary to select the most suitable combination using quality of service parameters. The problem of what extent the process must be done automatically remains the subject of many of those who work in this field.

To identify and explore existing services in the domain of configuration management we should consider activities, tasks, types of configurations and other items. Accordingly, we can classify services and the related sub-services in accordance with Table 1. Then combine the identified services in this domain. A directed graph called the dependency graph is used to store information of incoming and outgoing services. In this graph, there are two different types of nodes: data nodes and service nodes. Service nodes represent services registered in the service repository and data nodes are used to display the types of data that are produce by the service as output or for performing the services needed as inputs. Also edges of the graph are used to represent dependencies, see Figure 4.

Using information from the service composition we assume that given a service request the model will receive *Type of change* input data and produce *Create version diagram* output data.

Initially, we need a way to intuitively show that each service in the dependency graph produces a value in *Create version diagram* data.

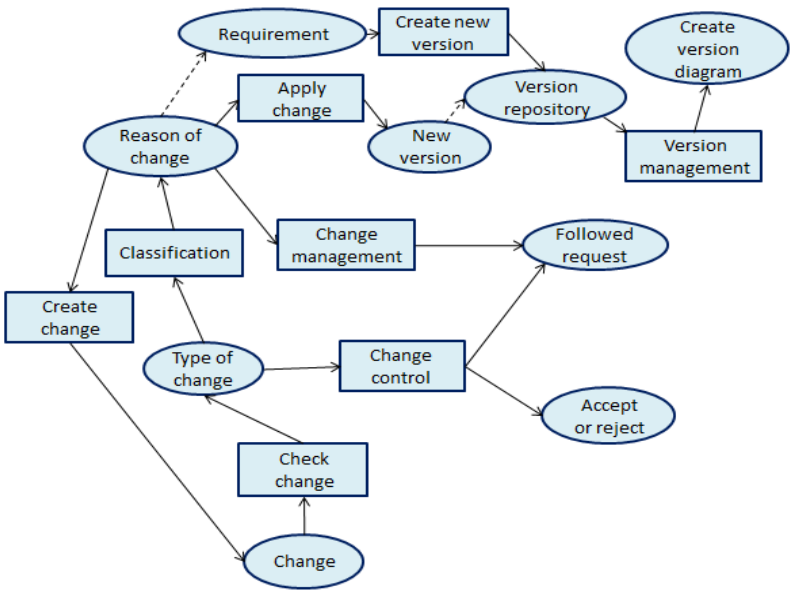


Figure 4. Dependency graph of the created change service

For comparison we use a numerical method of valuation. First, given that the value of a node of data type is equal to one value which the user has requested as output and the given value of the other nodes are equal to zero. Then explore at one loop for each service if the total value of service outputs divided by 2 is greater than the current value of the service, the service will be changed to Formula (1) as below:

$$NewValue_S = \max\{OldValue_S, \frac{\sum_{(S,O_i)\in OE} Value_{O_i}}{2}\} \quad (1)$$

If the value of a service changes during the process it is necessary to update the value of the input data associated with these services. In this case the value of data type will be equal to the maximum current value and half the value of the service provided.

In addition to the mentioned updates, if the value of a node of data type is changed, the value of other nodes of data type that are connected by edges of the cutting line will be changed to the maximum current value of this node. These updates will continue until the value of all the nodes in the dependency graph is fixed and there is no other change. Figure 5 shows the values of different nodes in the *Create version diagram* output produced.

Change management and *Change control* services have no value in the production of the output and thus will not be included in the desired composition.

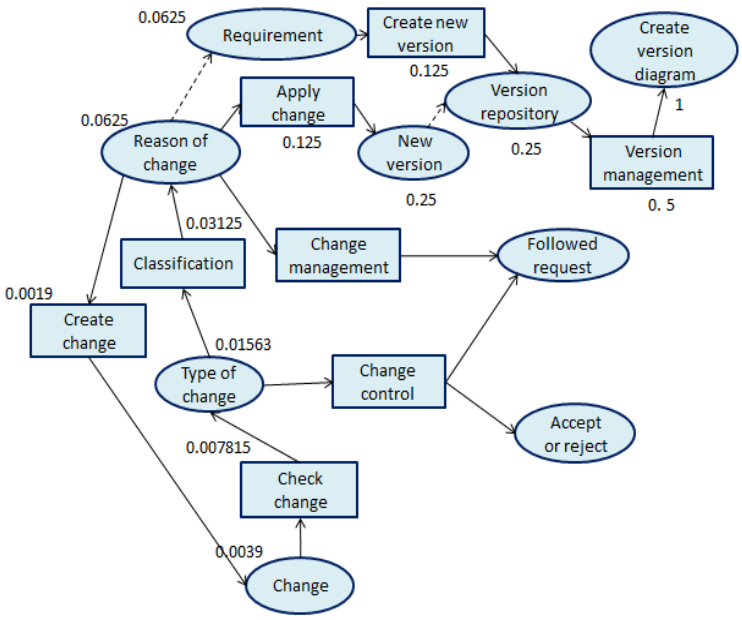


Figure 5. The value of nodes of dependency graph in the produced Create version diagram

We need to see which services are worth more than zero in the output produced and which one of these will be placed in composition structure. For this purpose we must find a way to move from inputs to outputs. In the first phase, check which services are applicable and obtain their input.

Regarding our example in the first step, we possess data type *"Type of change" so Classification* and *Check change* services are applicable. As another service may have a higher value of

Classification services, these services are added to the composition and its output is added to the list of available data types. In addition, all data types of its parent are added to the list for each data type. Similarly, in the next stage, we are looking for the most valuable service that all input data types are available and they have not already been used. Figure 6 shows the structure of the produce composition.

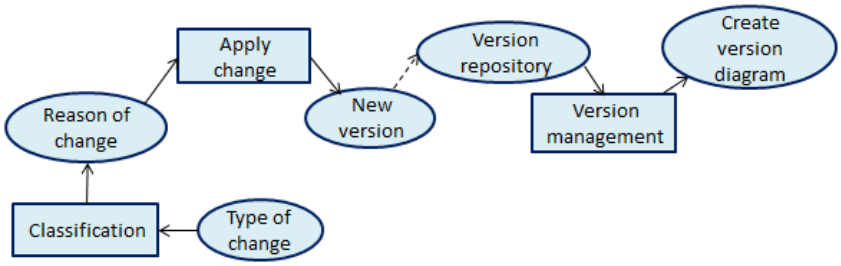


Figure 6. Composition of the Create version diagram Output from the Type of change input

Table 1. Services and Sub-services of Configuration Management.

Services	Sub-services of services	Subservices of sub-services
Service for identification configuration	Service to explore communications among CIs	Service to identification CIs
		Service for applying criteria of selection such as risks, technology, etc
	Service for access to configuration items	Service for providing information include requirements, documents, etc
		Service for communicate information
	Service for enumeration CIs	-
	Service to create product baseline	Service for information gathering related to product configuration
Service for control of configuration changes	Service to identify and documentation necessity of Change	Service to explore details of CIs
		Service to describe proposed change
		Service to explore reasons to create change
		Service to explore type of change
	Service for evaluation of change	Service to explore technical advantages of the proposed change
		Service to explore risks related change

Services	Sub-services of services	Subservices of sub-services
		Service to explore effect change on scheduling and costs
	Service to explore status change	-
	Service for implementation and release of change	-
	Service for validation of change	-
	Service for version management	Service for composite methods and tools
Service for status accounting configuration	Service to record changes in CIs	-
	Service for identification information of status configuration	-
	Service for maintenance history related to status CIs	-
	Service for tracking change	Service for support tools and processes
	Service for reporting status configuration	-
Service for audit configuration	Service for validation, tracking, and proper implementation of changes	-
	Service for measuring impact of configuration management process	-
	Service for guaranteed proper management of CIs	-
	Service for identification information to review	-
	Service for analysis and documenting the revised results	-
Service for reconfiguration	Remove service	Service for identification components, connectors and interfaces
	Overlap service	Service for identification of components, connectors and interfaces
	Insert service	Service for identification of components, connectors and interfaces
	Move service	Service for identification of components, connectors and interfaces

Services	Sub-services of services	Subservices of sub-services
	Replace service	Service for identification components, connectors and interfaces
	Implode service	Service for identification of components, connectors and interfaces

5. COMPARISON

In this section, we compare a number of previous works done in the field of configuration management software architecture and our proposed solution based on quality attributes. Solution 1 (S1) stands for the Mae environment, solution 2 (S2) denotes the Molhado approach and solution 3 (S3) is the Ragnarok model.

Table 2. KPI for Solutions.

Quality Indicator	Solution 1	Solution 2	Solution 3	Our solution
Run-time support	0.441	0.343	0.439	0.484
Flexibility	0.352	0.434	0.272	0.566
Variability	0.522	0.250	0.400	0.411
Completeness	0.537	0.496	0.496	0.666
Tool Support	0.640	0.375	0.401	0.555

Table 2 shows solution candidates for different indicators and Table 3 ranks which candidate solutions are best for each quality attribute. These two tables were obtained from an experimental study by seven domain experts with more than ten years of experience in this field [13].

Table 3. Solution in KPI.

Quality Indicator	Solution 1	Solution 2	Solution 3	Our Solution
Run-time support	0.711	0.706	0.550	0.729
Flexibility	0.765	0.543	0.388	0.649
Variability	0.695	0.555	0.333	0.588
Completeness	0.617	0.765	0.500	0.649
Tool Support	0.473	0.578	0.732	0.416

Table 2 was constructed from a pairwise comparison of solution candidates for all quality attributes and Table 3 shows each quality attribute for all candidate solutions. Colored cells in the table indicate how the comparison has been done. Pairwise comparisons were calculated with the

Analytic Hierarchy Process approach (AHP) by taking advantage of the scales mentioned in Table 4.

Table 4. Scale for Pairwise Comparison using AHP [25].

Relative Intensity	Definition
1	Of equal importance
3	Slightly more important
5	Highly more important
7	Very highly more important
9	Extremely more important

Run-time of our solution was more than both S1 and S2. This improvement is due to the use of service-oriented approach. Also, the flexibility in our solution was better than the solutions in Table 2 but only better than S2 and S3 from Table 3. As a result we can conclude that this indicator is better than S2 and S3; however, we can't conclude it is better than S1. Variability of our solution is better than S2 and S3 but is worse than S1. Completeness of our solution is better than S1 and S3 but we not S2. Finally, tool support in our approach is worse than S1 but approximately the same as S2 and S3. According to the quality attribute results obtained in this study the effect of the service-oriented approach produced an improvement in the configuration of software architecture.

6. CONCLUSIONS

Software configuration management has the potential to managing evolution and control changes in software systems. In fact, software architecture is one of the most challenging issues in the maintenance phase of very large systems. In this paper we reviewed a variety of previous works and existing challenges in this field, and also examined weaknesses associated with these works. Hence, in this paper an architectural model has been presented that takes advantage of existing concepts and mechanisms in the configuration management field, such as versioning, composition, consistency and construction; it also uses the concepts of the service oriented approach such as loose coupling, abstraction, reuse, heritability, autonomy, etc. The model then applies these two ideas to construct a better configuration to optimize the software architecture. It also provides the possibility for a composite of services and improvement of the configuration of the software architecture.

Generally, the model reduces total cost. In the future, we will try to improve our view by utilizing mechanisms base on the enterprise service bus concept.

REFERENCES

[1] Westfechtel, B., & Conradi, R. Software architecture and softwareconfiguration management. In Software Configuration Management,pp. 24-39, 2003.

[2] Santiago, I., Vara, J. M., Verde, J., de Castro, V., & Marcos, E.Supporting Service Versioning-MDE to the Rescue. In ENASE, pp. 212-217, 2013.

[3] Nguyen, T. N., Munson, E. V., Boyland, J. T., & Thao, C. Architecturalsoftware configuration management in Molhado. In SoftwareMaintenance, 2004. Proceedings. 20th IEEE International Conference,pp. 296-305, 2004.
[4] Cândido, G., Colombo, A. W., Barata, J., & Jammes, F. Service-orientedinfrastructure to support the deployment of evolvable productionsystems. Industrial Informatics, IEEE Transactions on, 7(4), pp. 759-767, 2011.
[5] Balderrama, J. R., Montagnat, J., & Lingrand, D. jGASW: a service-oriented framework supporting HTC and non-functional concerns.InWeb Services (ICWS), 2010 IEEE International Conference, pp. 691-694, 2010.
[6] Jones, M., & Hamlen, K. W. A service-oriented approach to mobilecode security. Procedia Computer Science, pp.531-538, 2011.
[7] van der Hoek, A., Mikic-Rakic, M., Roshandel, R., & Medvidovic, N.Taming architectural evolution. ACM SIGSOFT Software EngineeringNotes, 26(5), pp. 1-10, 2001.
[8] Christensen, H. B. The Ragnarok Architectural Software ConfigurationManagement Model. In Systems Sciences, 1999. HICSS-32.Proceedings of the 32nd Annual Hawaii International Conferenceon IEEE, pp. 7, 1999.
[9] Estublier, J. Software configuration management: a roadmap. InProceedings of the Conference on the Future of Software Engineering,pp. 279-289, 2000.
[10] Lacy, S., & Norfolk, D. Configuration Management: Expert guidancefor IT service managers and practitioners-Revised edition. BCS, 2014.
[11] Hardion, V., Spruce, D. P., Lindberg, M., Otero, A. M., Lidon-Simon, J.,Jamroz, J. J., & Persson, A. Configuration Management of the controlsystem. THPPC013, 2013.
[12] Pressman, R. S., & Lowe, D. B. Web engineering: a practitioner'sapproach. McGraw-Hill Higher Education, 2009.
[13] Erl, T. Service-oriented architecture: concepts, technology, and design.Pearson Education India, 2005.
[14] Lublinsky, B. Defining SOA as an archi-tecture style.IBM DeveloperWorks, 2007.
[15] Georgas, J.C., van der Hoek, A., Taylor,R.N.Architectural RuntimeConfiguration Management in Support ofDependable Self-AdaptiveSoftware. In: Workshop onArchitecting Dependable Systems, 2005.
[16] Eyermann, F., Racz, P., Stiller, B., Schaefer, C., & Walter, T. Service-oriented accounting configuration management based on diameter.In Local Computer Networks, 2005. 30th Anniversary. The IEEEConference ,pp. 621-623, 2005.
[17] Nguyen, T.N., Munson, E.V. andBoyland,J.T. The molhado hypertextversioning system. In Conferenceon Hypertext and Hypermedia,pp.185-194, 2004.
[18] Nguyen, T. N.,Munson, E. V., & Thao, C. Object-oriented configurationmanagement technologycanimprove software architecturaltraceability. In Soft-wareEngineering Research, Management andApplications, 2005.Third ACIS International Conference, pp. 86-93,2005.
[19] Nguyen, T. N., Munson, E. V., Boyland, J., & Thao, C. An Infrastructurefor Development of Multi-level, Object-Oriented ConfigurationManagement Services. In Proceedings of the 27th InternationalConference on Software Engineering, pp. 215-224, 2005.
[20] Nguyen, T. N., Munson, E. V., Boyland, J. T., & Thao, C. Multi-levelconfiguration management with fine-grained logical units. In SoftwareEngineering and Advanced Applications, 2005. 31st EUROMICROConference, pp. 248-255, 2005.
[21] Oreizy, P., Medvidovic, N., & Taylor, R. N. Architecture-based runtimesoftwareevolution. InProceedings of the 20th internationalconferenceon Softwareengineering, pp. 177-186, 1998.
[22] Nguyen, T. N., Munson, E. V., Boyland,J. T., &Thao, C. Multi-levelconfigurationmanagement with fine-grained logical units. In SoftwareEngineering and AdvancedApplications, 2005.31st EUROMICROConference, pp. 248-255, 2005.
[23] Pizzi, S. V. A routing architecture forthe airborne network. In MilitaryCommunications Conference, pp. 1-7, 2007.
[24] Ran, S. A model for web services discoverywith QoS. ACM Sigecomexchanges, pp. 1-10, 2003.
[25] Svahnberg, M., & Wohlin, C. A Comparative Study of Quantitative andQualitative Views of Software Architectures. In Proceedings of the 7thInternational Conference on Empirical Assessment in SoftwareEngineering, pp. 1-8, 2003.

Authors

Razie Alidoosti received her MS degree in computer engineering from Sharif University of Technology, Tehran, Iran. She is currently a Research Assistant at Sharif University of Technology. Her research interests include Software Architecture, multi agent systems, machine learning, Analyze Method. She can be contacted at Performance Evaluation Software Engineering Lab, No. 601, Computer Engineering Department, Sharif University of Technology; Alidoosti@ce.sharif.edu.

Shahrouz Moaven is a PhD candidate at Sharif University of Technology from 2009. He received his MS degree in Software Engineering at Sharif University of Technology, Tehran, Iran. His research interests include software architecture, software engineering, decision support system and business intelligent. He can be contacted at Performance Evaluation Software Engineering Lab, No. 601, Computer Engineering Department, Sharif University of Technology; moaven@ce.sharif.edu.

Jafar Habibi received PhD degree in computer science from University of Manchester in 1998. He is currently an associate professor and chairman of Computer Engineering Department at the Sharif University of Technology, where he has been a faculty member since 1989. His research interests include software engineering, software architecture and design, performance evaluation, system analysis and design, information systems and simulation. He can be contacted at No. 626, Department of Computer Engineering, Sharif University of Technology; jhabibi@sharif.edu.