Dynamic Web Services Composition Using Change Management System

B. Muruganantham[1], K. Vivekanandan[2] P. Kirubanantham[3]

[1] Asst. Professor, [2] Professor, [3] M.Tech Student,
[1,3]Department of computer science and Engineering, SRM University, Chennai, India.
[2]Department of computer science and Engineering - Pondicherry Engineering College, Pondicherry, India
[1]muruganantham.b@ktr.srmuniv.ac.in, [2]k.vivekanandan@pec.edu, [3]kirubanantham04@gmail.com

***Abstract*— A Long-term Composition Service (LCS) is a dynamic collaboration between autonomous web services that collectively provide a value-added service. LCS has a long-term commitment to its users. Earlier technologies proposed various integrated framework that are managed changes of long term composed services. The frameworks are also used for some techniques to handle the change reaction of LCS. But, the earlier frameworks cannot be effectively process the most challenging issues of change management such as how to automate the process of change reaction. To overcome this issue, evolution of Long-term Composed Services (Ev-LCS) has been proposed. This system provides the grounding semantics to support the automation of change management and it is offered a set of change operators that is allowed to specify a change in a specific and formal manner. After that, it is proposed a change enactment strategy that actually implements the changes. Finally, develop a prototype system such as travel agency for the proposed Ev-LCS to demonstrate its effectiveness. Further, enhance our proposed system that works on multiple concurrent changes in the LCS.**

Keywords—Long-term Composition Service, Change Management & Prototype.

I. Introduction

Service-Oriented Computing (SOC) is emerging as a new paradigm to facilitate functionality outsourcing on the web. This is enabling a paradigm shift in business structures allowing them to outsource required functionality from third party web based providers through service composition. A composed web service is, therefore, on-demand and dynamic collaboration between autonomous web services that collectively provide a value added service. Each autonomous service specializes in a core competency, which reduces cost with increased quality and efficiency for the business entity and its consumers. There are two types of composed web services: short term and long term. A short-term composed service has a very limited lifetime. It is usually generated to fulfill a specific business goal for a limited number of users. Once the goal is reached, the collaboration among its component services is then dissolved. A long-term composed service (LCS), in contrast, has an open-ended lifetime. It usually has a long-run business goal and business commitment to its customers. The partnership among its component services is relatively stable unless the occurrence of some exceptional event. LCSs have attracted a lot of attentions since they empower a virtual enterprise, which is a cross-organization collaboration to offer value-added and customized services. Fully realizing an LCS lies in providing support to improve its adaptability to the dynamic environment, i.e., to deal with changes during its lifetime. Because of the dynamic nature of web service infrastructure, multi changes should be considered as the rule and managed in a structured and systematic way. Changes are usually introduced by the occurrence of new market interests, business regulations, new technologies, and so forth. Such multi changes are always associated with a requirement on the modification of an LCS with respect to the functionality it provides, the way it performs, the partners it is composed of, and the performance it delivers. Once a change occurs, an LCS needs to quickly adjust itself to fulfill the requirement introduced by the change. The adjustment also needs to be performed in an automatic manner considering the frequent occurrence of changes. By doing this, an LCS can maximize the market interests it attracts, optimize the way it outsources its functionality, and thus maintain its competitiveness among its peers.

Changes in an LCS can be classified into two categories: top-down changes and bottom-up changes. Top-down changes refer to those that are initiated by the owner of an LCS. These are usually the result of new business requirements, new regulations, or new laws. For example, the owner of an LCS may want to add a new functionality to attract more customers. Bottom-up changes refer to those that are initiated by the outsourced web service providers. For example, an airline reservation service provider may change the functionality of the service by adding a new operation for checking flight status, or a traffic service provider may decide to increase the invocation fee of the service. In this project, we focus on dealing with top-down changes. Change management in LCSs poses a set of research issues. An LCS outsources its functionality from independent service providers. There are no central control mechanisms that can be used to monitor and manage these service providers. Therefore, the challenge of managing changes lies in providing an end-to-end framework to introduce, model, and manage a top-down change in a way that best reacts to the change. More specifically, changes need to be first captured and modeled in a formal way so that they can be understood and processed. After that, an efficient and complete change reaction process needs to be designed and implemented. Finally, the change reaction process needs to be verified to ensure the correctness of an LCS. We expect that changes in an LCS occur frequently due to the dynamic business environment it interacts with. Thus, it is not practical to manage all changes in a manual way. Change management has been a major research topic in the area of workflow management. There are many frameworks proposed for supporting adaptive workflow systems. These works mainly address the issue of handling the running instances when the schemata have been modified. The major research issues of change management in LCSs are different from the ones in workflow systems. This is due to the different contexts and assumptions. First, LCSs have the promise of enabling flexible collaboration between different business entities. They should be capable of adapting to the dynamic environment in a prompt way. In this context, the modification of the collaboration is expected to be frequent and requires a systematic support. In contrast, the collaboration between different entities of workflow systems is usually within an organization. The environment that the systems interact with is relatively "static." Changes on the schemata are treated as "exceptions" and usually handled in a manual way. Second, LCSs outsource their functionalities from autonomous service providers. There is not a central control mechanism that an LCS can rely on to monitor and manage changes. In contrast, the entities in workflow systems are within an organization, which enables a centralized approach to monitor and manage changes. Therefore, change management in LCSs has different focuses from the ones in adaptive workflow systems. It is not sufficient to manage changes in an LCS by simply applying the approaches adopted in existing frameworks. Moreover, we assume that the LCS operates in rich and complex environments. This is typified by the web. The process of managing top-down changes is affected by various factors, including the current structure of an LCS, the type of a change, the services that are involved in the change, the relationships between component services, etc. It is usually hard to predict what a change will be, what the triggered "side effects" of this change are, etc. Therefore, it is infeasible in this context to predetermine how to deal with a change before its occurrence. It is also not practical to pre-design different change management processes for different LCS structures.

This is different from traditional rule-based systems, where a set of rules are predefined to make decisions under expected situations. In this project, we propose an end-to-end framework to manage top-down changes in LCSs. We design machine-process able semantics and leverage the semantics to manage changes on the fly.

II. RELATED WORK

Change management is an active research topic in database management, Information engineering, and software development. Explore efforts are also underway to provide change management in the web service community and adaptive workflow systems [2], [8]. In this section, we intricate on some representative work and distinguish them from our work. In [2], the work focuses on organizing bottom-up changes in service-oriented enterprises. Changes are eminent between service stage and business stage: triggering changes that taken place at the service level and hasty changes that occur at the business level in response to the triggering changes. A set of mapping regulations are defined between triggering changes and reactive changes. These rules are used for promulgating changes. A Petri net-based change model is proposed as a means for automatically reacting changes. Agents are employed to help out in detecting and managing changes to the enterprises. In [13], a technique is presented based on decoupling the migration algorithm with the internal structures of workflow types of modification operations. They combined workflow type versioning with workflow instance migration to support dynamic changes on executing instances in a flexible manner on demand. In [4], a framework is presented to detect and react to the exceptional changes that can be raised inside a workflow-driven web application.

It first classifies these changes into behavioral (or user-generated), semantic (or application), and system exceptions. It then proposed a modeling framework that describes the structure of activities inside hypertexts of a web application. The hypertext belonging to an activity is broken down into pages, which are identified within an activity. It proposes a framework to handle these changes. Compared with the works in [2] and [4], our work focuses on dealing with a different type of changes, which are top down changes that occur in a composed service. In [6], the work proposed a MIL (Minimal Latency) -based approach for providing best QoS in Internet based web applications. They uses geographical replication approach to face the quality of service issue in web services, for that it used client side approach rather than server side approach. It reduces more overhead and traffic congestion through the network, but it cannot analyze the other part of replication based approach that is cluster of servers. In [6], work mainly focus on the client side requests with static management of services, which is different from our work. In [20], the work proposes a framework that manages the business protocol evolution in a service-oriented architecture. It uses several features to handle the running instances under the old protocol. These features include impact analysis and data mining-based migration analysis. The impact analysis is to analyze how protocol change impacts the running instances. It will be used to determine whether ongoing conversations are changeable to the new protocol or not.

The data mining-based migration analysis is used for cases where the regular impact analysis cannot be performed. Service interaction logs are analyzed using data mining techniques. It then uses the result of the analysis to determine whether a conversion is migrating able or not. In [20], the work mainly focuses on maintaining the consistency between business protocols and their running instances, which is different from the focus of our work. In [11], a model-driven methodology is proposed for the development of composite context-aware web services. It models context-aware services to provide personalized services to users. The modeling profiles include a web service profile and a context metamodel. The context metamodel can be used to model the context information. Context binding is used to model the association between a piece of context information and context-aware objects, such as messages and operations. Context triggering is used to define a set of context adaption policies, guiding the service selection and invocation based on the context information. Although it is not discussed in [11], the model-driven methodology can be also used for change management. That is, some changes can be modeled as context and the corresponding change reaction policies can be predefined in context triggering. Nevertheless, it mainly focuses on how to model context-aware composite services and translate the high-level model (i.e., UML diagram) to low-level executable codes. In contrast, we focus on modeling different types of top-down changes in an LCS and implementing these changes in an automatic way.

In [8], the work focuses on modeling dynamic changes within workflow systems. It introduces a Modeling Language to support Dynamic Evolution within Workflow System (ML-DEWS). A change is modeled as a process class, which contains the information of roll-out time, expiration time, change filter, and migration process. The rollout time indicates when the change begins. The expiration time indicates when the change ends. The change filter specifies the old cases that are allowed to migrate to the new procedure. The migration process specifies how the filtered in old cases migrate to the new process. The new version of the workflow schema is predefined. In our work, the new LCS schema is automatically generated. We leverage the semantics from the service ontology and the LCS schema to support the automation of change management process. In [14], Hierarchically-structured web service ontology is proposed. It suggested a heuristic approach for query using ontology and tree coloring. But it does not provide an effective structure to construct the process graph and not clearly mentioned about selecting next web service when change is required. In [11], it uses semantic web services in managing manufacture processes, in which they used OWL-S for composition of web services. They took three set of services in which one of the web service maintain a semantic model of the in progress state of the system, while remaining uses the model to compose the domain web services. But their work was complex and time consuming because each time semantic property will check on arrival of new service, another problem is scalability.

A. Limitations of Existing System

1. In bottom up, Agents are employed to assist in detecting and managing changes to the enterprises.

2. In event-based frame work, by associating a change with an event, the change on one or more member services of a Long-term Composition Service can be propagated to the other Long-term Composition Service participants.

3. Event-based framework is not for change management purpose.

4. Event-based frame work does not support for dealing with top-down changes at all, which are initiated by the dynamic business environment, not by the individual services.

III. PROPOSED ARCHITECTURE

The proposed travel agent system provides a graphic user interface for users to specify two types of information: an LCS schema and a change specification. A change specification contains the required information for the purpose of change management. The process and result of change management will be returned to users via the user interface. The change management module is the kernel component of the system. The process of change reaction and change verification will be traced and visualized by a visualization module. The change management module relies on two supporting components, including web service providers (Airline-(International, National, and Local), Taxi, Rental car, hotel, weather report web service, and payment web service) and ontology providers (Semantic Description of common features of web services). The web service providers offer the actual web services. The ontology providers offer the formal and sufficient semantics to support the automation of change management. The proposed system architecture for web service composition is shown in the Fig 1. It has following advantages. They are,

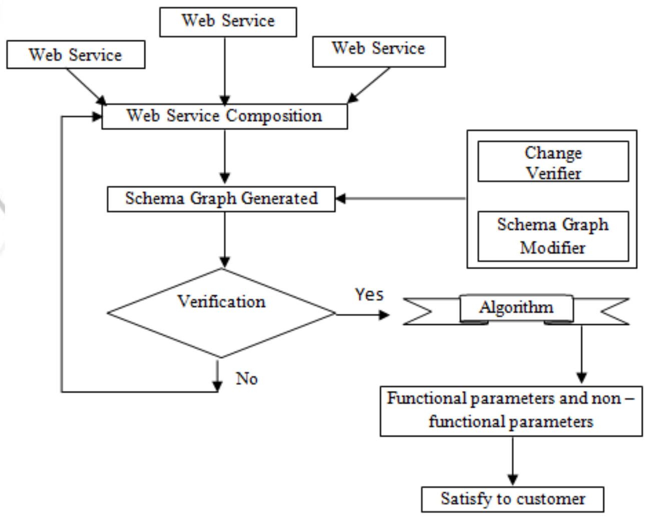

Fig 1Proposed Architecture for web service composition

1. Proposed System automatically handles the multiple changes in the long-term web services
2. But, it does not reduce the proposed system performance when compared with earlier techniques.
3. Mainly accuracy increased.
4. The new various web services are added and composed effectively.
5. The new web services adding does affect the control flow and data flow of proposed system.

Finally, complete content and organizational editing before formatting. Please take note of the following items when proofreading spelling and grammar:

Functional Requirements

This system is done for handle a user specified change in web services that are automatically reprocess without any interference.

- **Input:** The input of this project is a user specified multiple changes that are provide to travel agency
- **Behavior:** The behavior this project is constructed the modify schema graph using user specified changes and change operators. Then the schema graph is verified by graph verifier. The web services are accessed using that modified schema graph.
- **Output:** The output of this project is travel ticket information shows from travel agency to requested user.

Non Functional Requirements

Performance: Time spending of schema graph update is not depended on the LCS size

Performance increased based on the change affected node

Usability: This project is using server and client. Server will be used for forwarding the user indicate change based service results. This system used effectively in the change environment. LCS Reliability: The reliability of this project is very high.

IV. IMPLENENTATION AND RESULTS

The process of orchestrating the selected services together follows the service composition defined in the updated LCS schema graph. It consists of two steps. First, it needs to map the service composition in a service level view (i.e., global view) to an operation level view (i.e., local view). Second, the operation level service composition will be translated to an executable BPEL (Business Process Execution Language) description.

Each service s in an LCS schema, we maintain an operation execution graph, where the node set s.OP includes the operations that will be actually invoked, and the edge set s.E that defines the execution order among the operations. s.OP and s.E are initially empty. A set of key operations will be first selected and included in s.OP to generate the desired service output. A key operation refers to the one that generates a subset of service output. After that, for each operation in s.OP, we use operation-level dependence to determine its depending operations and include them in s.OP. We also add the corresponding edges to s.E, showing that the operation should be invoked after the invocation of its depending operations. This process continues till all the operations in s.OP and their depending operations are also in s.OP. The operation execution graph is then generated, which provides sufficient information for invoking the service.

Schema graph is generated for the composition of web services can be added or removed, automatically changes will be made and it will not affect the composition.

Performance

The performance of the composed web services are measured with the help of response time. Here 5 users and nearly 40 services are taken for composition. Depends on the user requirements, the services are fetched and composed the corresponding response time is calculated is shown in the fig 2.

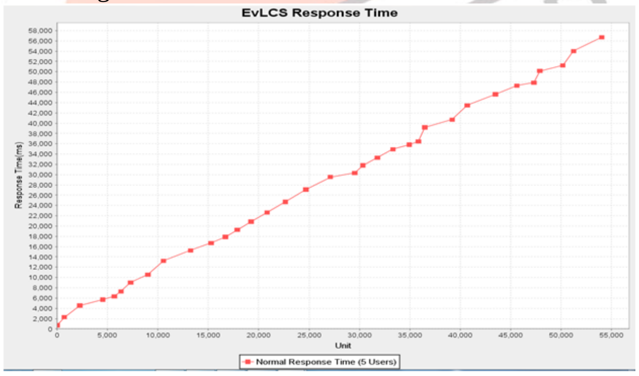

Fig 2 Ev-LCS Response Time.

The services added or removed during composition and the corresponding change response time is calculated is shown in the fig 3.

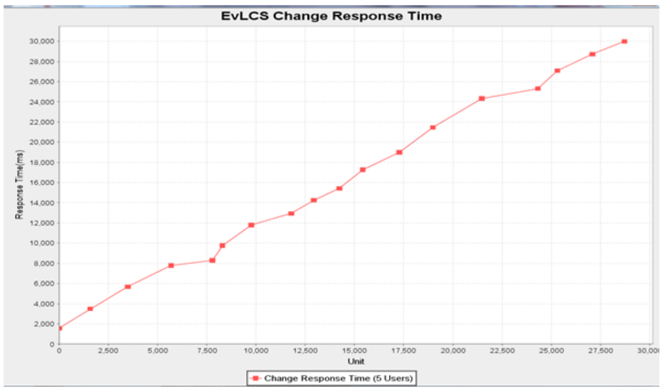

Fig 3 Ev-LCS Change Response Time.

CONCLUSION

This project proposed an Ev-LCS, an end-to-end framework that specifies, reacts to, and verifies top-down changes in an LCS. A formal model to provide the grounding semantics to support the automation of multi change management, including web service ontology and an LCS schema has been proposed. It then defined a set of change operators to specify top-down multi changes based on the formal model. For the change operators, that proposed a set of algorithms to automatically implement them. The correctness of the LCS is also ensured during the process of change enactment. It implemented a prototype of the proposed change enactment process to prove its practicality. In future work, implement a machine learning approaches to manage the multiple changes.

REFERENCES

[1] X. Liu, C. Liu, M. Rege, and A. Bouguettaya, “Semantic Support for Adaptive Long Term Composed Services,” Proc. IEEE Int’l Conf. Web Services (ICWS ’10), July 2010.

[2] G.M. Kapitsaki, D.A. Kateros, G.N. Prezerakos, and I.S. Venieris, “Model-Driven Development of Composite Context-Aware Web Applications,” Information and Software Technology, vol. 51, pp. 1244-1260, Aug. 2009.

[3] Q. Yu, X. Liu, A. Bouguettaya, and B. Medjahed, “Deploying and Managing Web Services: Issues, Solutions, and Directions,” VLDB J., vol. 17, pp. 537-572, May 2008.

[4] S.H. Ryu, F. Casati, H. Skogsrud, B. Benatallah, and R. Saint-Paul, “Supporting the Dynamic Evolution of Web Service Protocols in Service-Oriented Architectures,” ACM Trans. Web, vol. 2, no. 2, pp. 1-46, 2008.

[5] Y. Baghdadi, “A Web Services-Based Business Interactions Manager to Support Electronic Commerce Applications,” Proc. Seventh Int’l Conf. Electronic Commerce (ICEC ’05), pp. 435-445, 2005.

[6] M.S. Akram, B. Medjahed, and A. Bouguettaya, “Supporting Dynamic Changes in Web Service Environments,” Proc. Conf. Service Oriented Computing (SOC), 2003.

[7] C.A. Ellis and K. Keddara, “A Workflow Change is a Workflow,” Proc. Business Process Management, Models, Techniques, and Empirical Studies Conf., pp. 201-217, 2000.

[8] M. Brambilla, S. Ceri, S. Comai, and C. Tziviskou, “Exception Handling in Workflow-Driven Web Applications,” Proc. 14th Int’l Conf. World Wide Web (WWW ’05), 2005.

[9] M. Kradolfer and A. Geppert, “Dynamic Workflow SchemaEvolution Based on Workflow Type Versioning and Workflow Migration,” Proc. Conf. Cooperative Information Systems, pp. 104-114, 1999.

[10] M. Conti, M. Kumar, S.K. Das, and B.A. Shirazi, “Quality of Service Issues in Internet Web Services,” IEEE Trans. Computers, vol. 51, no. 6, pp. 593-594, June 2002.

[11] Juha Puttonen, Andrei Lobov and José L. Martinez Lastra, " Semantics-Based Composition of Factory Automation Processes Encapsulated by Web Services ", IEEE Transactions On Industrial Informatics, Vol. 9, No. 4, pp. 2349-2359, November 2013.