Journal of Computer Science 10 (10): 1946-1954, 2014
ISSN: 1549-3636

doi:10.3844/jcssp.2014.1946.1954 Published Online 10 (10) 2014 (http://www.thescipub.com/jcs.toc)

OPTIMIZED INCIDENT MATCHING AND AUTO-MATED VERIFICATION OF COMPOSITION PAT-TERNS IN LONG TERM COMPOSED SERVICES

[1]Thirumaran, M., [2]M. Jannani and [3]P. Dhavachelvan

[1,2]Department of Computer Science and Engg, Pondicherry Engineering College, Puducherry, India
[3]School of Engineering, Pondicherry University, Puducherry, India

Received 2014-02-05; Revised 2014-02-10; Accepted 2014-05-12

ABSTRACT

With the increase in the need and demand for evolving Web services, the rate at which changes are made to the services has increased. In case of importunate change requests, there arise critical situations where the business analysts are subjected to make changes by themselves without the aid of the developers because of the time and cost factors. However, there are high chances that an analyst makes a bug introducing change and injects incorrect statements into the logic and hence there are fair chances for the changed service to exhibit an undesired behavior. Though the impact of the changes is analyzed and recorded every time a change is made and the bug report is generated, it is often done many months after the initial injection of the bug which is time consuming and ultimately results in the failure to meet the business outcome. This process is repeated even when similar change requests are encountered which is absurd in the current scenario and acts as a challenge to the success of a business which is the motivation behind this study. This study address this challenge by focusing on an efficient prediction system which would analyze the recorded incidents, filter the incidents which match with the current incident and predict the level of risk and accuracy involved in committing the change. The implication is that the system performs automated verification of composition patterns and detection of violations in the business policies if any and aids in change management.

Keywords: Service Oriented Architecture, Web Services, Web Service Change Management, Incident Matching, Service Composition

1. INTRODUCTION

Consider a business analyst working with Web services in long term composition. It is more likely for situations which demand making changes immediately without waiting for the software development team. An analyst with very little knowledge over the code and remarkable knowledge over the domain is likely to make a bug-introducing change leading to latent issues and eventually to the business failure since bugs in the source code result in undesired external behavior especially when the change involves the addition or substitution of a required service. So when the available choices for service replacement are encountered, there arises the necessity to check if the choice matches the need. When the analyst is put in a situation to analyze the services and find out the best choice by verifying the composition pattern, it consumes a considerable amount of time. This study presents a system which would enable the verification of the composed services automatically and would enable the analysts to take steps to make the changes immediately and efficiently. The system enables prediction of the level of accuracy and risk involved by estimation of the detection rate, degree of automation and error rate. This automated verification of composition patterns facilitates efficient incident matching and extraction of highly similar incidents.

Corresponding Author: Thirumaran, M., Department of Computer Science and Engg, Pondicherry Engineering College, Puducherry, India

This holds promise for reducing the time required to make changes to services in long term composition.

2. RELATED WORKS

The works pertinent to the focus of our paper are elucidated in this section. Todd *et al*. (2000) have defined code to be aged or decayed if its structure makes it unnecessarily difficult to understand or change and we measure the extent of decay by counting the number of faults in code in a period of time. This involves predicting false incidents based on software change history whereas in our work, false incidents refer to the change incidents which were not successful. A change propagation model to predict the change propagation on the downstream activities due to different degrees of change that might be initiated at different stages during a design project has been proposed by (David *et al*., 2012). Architecture with a QoS based web service clustering method which helps us to select the best service that suits user quality preferences has been propounded by (Wu *et al*., 2009). Liu *et al*. (2013) have proposed a change management framework where managing changes in Long term Composed Services (LCS) which deals with both functional and non functional change requests. But they have not proposed any methodology for performing emergency changes and they have not found the various problems that will be caused due to the sudden changes. Liu *et al*. (2011a) have presented a framework where managing changes in LCSs has been modeled as a dual service query optimization process. The optimization has been performed by considering only the non functional factors like reputation and the Quality of Service (QoS) factors. Akram *et al*. (2010) proposed an automatic change management framework that is based on the Petri net models which is used to manage triggering changes and reactive changes. Liu *et al*. (2011b) have also proposed ideas about the change management in semantic Web services. But they have implemented Petri nets for the analysis whereas in our work, we have implemented Finite State Machine for analysis. Plebani and Pernici (2009) have proposed an algorithm which combines the analysis of the Web Service Description Language (WSDL) structures and the analysis of the terms used inside them.

3. PROPOSED SYSTEM

The working of the proposed system which guarantees automated verification of composition patterns and optimized incident matching in long term composed services is elucidated in **Fig. 1**. Once the change request indicating the resource to which the change has to be made and the type of change to be made is obtained, the request handler performs the change request analysis which involves analyzing the change request with respect to domain and context in association with the planner and the domain analyzer. It analyzes and determines the change information such as where and when the request has originated, who has originated the request, what type of change is requested, which business logic is involved and what operation needs to be performed. The Planner automates the process of handling the request by diagnosing the nature of the request if it is a new one, if a similar request has been already handled or if the request is an already existing one. It also checks how often the changes occur and what the priority set for the change is. On account of a new change request, it proceeds to the next level with domain information and the BL Analyzer performs a rule bound analysis. This analysis makes sure that the extracted logic is bound to the set of rules. It checks if all segments of the extracted logic are bound with the rules available in Rule Set. This is followed by generating the schema by the Schema Generator which is XML based and uses tags for describing every rule, function, relationships etc. The schema generator constructs the LCS Schema. The schema comprises of reference points which carry some amount of knowledge in terms of Meta data in the BL schema. These reference points refer to historical events which help to map the existing change incidents with respect to the current change event. On event of a change, the reference points are fine-tuned and therefore the number of reference points will be gradually reduced. Reference points focus on the required business logic entity and the knowledge gathered increases gradually. Since all changes are done at the schema level by the business analyst, the evaluated change progress is notified as Meta data in the effect of any change event. The Property Evaluator act as a pre request to incorporate changes over the logic and checks whether the logic is computable, traceable, accessible and configurable in prior to make the actual change. The key goal of this Property Evaluator is to provide better run time support during the course of change progress. It in fact investigates the defects in the logic and validates the quality of business logic by automating various functional and non-functional assessments to ensure that new changes have not impaired existing functionality.

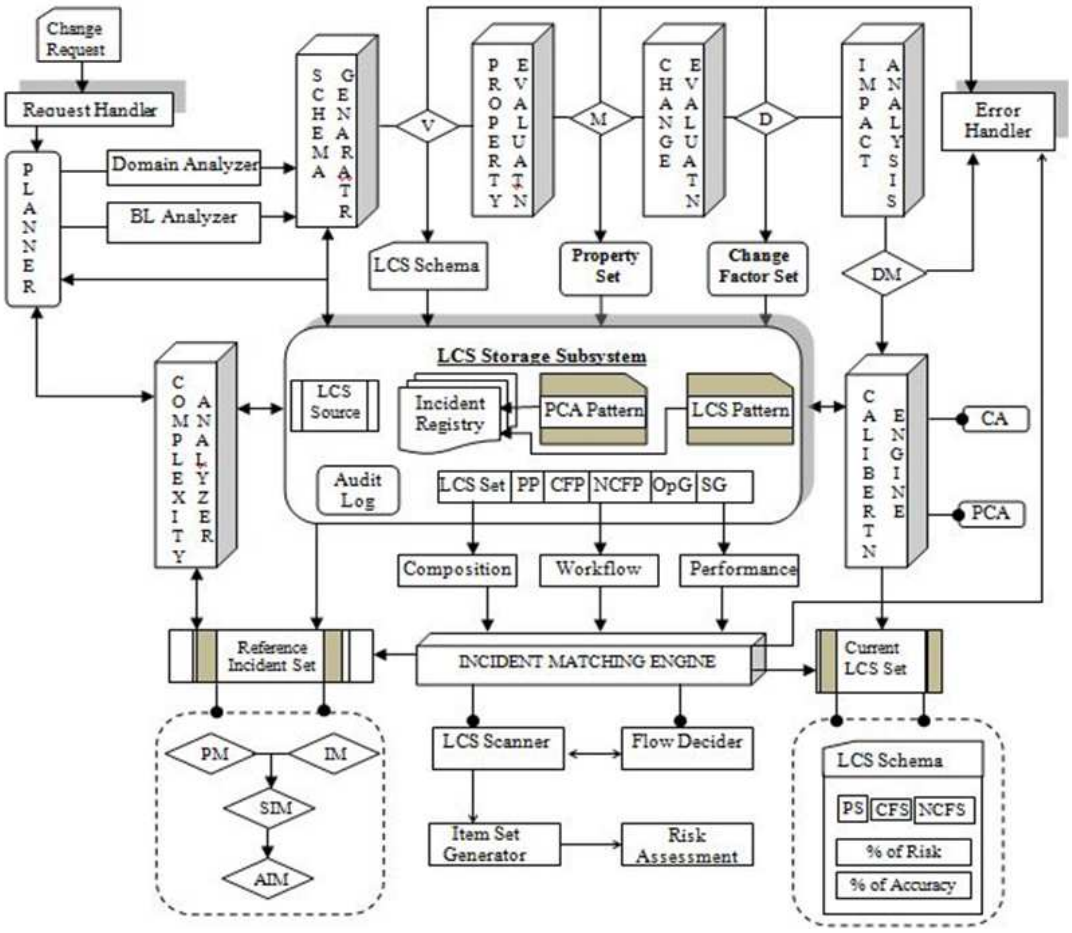

Fig. 1. Working of the proposed system performing incident matching

The results of the property evaluator are stored as the property set. The change evaluator helps in evaluating whether a change will be meaningful or not. This is very advantageous since whether a change is manageable or unmanageable is said well before. When all the run time support is available, a change is said to be manageable. Change evaluation provides efficient behavior analysis and provides the guidance verifying that the right path is followed. The results of the change evaluator are stored as the change factor set. With respect to impact analysis, the history of incidence matters. This is because there is no use of analyzing the impact when the request is fresh, occurring for the first time. The extent to which highly similar incidents are matched indicates the level of accuracy in impact analysis. This is the core functionality of the incident matching engine where the current incident LCS set is compared with the reference incident LCS sets. Whenever incident matching is done, for every change request, the current incident LCS set and the reference incident LCS sets differ. The reference incident LCS set is populated as a result of pattern matching, item set matching, similar item set matching and active item set matching. Since the LCS storage system act as a rich source of storage comprising of the LCS schema, incident registry, Probabilistic Cellular Automata (PCA) pattern, LCS pattern, Audit log, LCS set, property, change factor, non functional change factor patterns, the reference incident set is populated efficiently. The presence of rich source of reference has kindled the need for an efficient, optimized and automatic incident matching approach. The LCS scanner performs the vital duty of scanning the LCS sets, the flow decider functions for checking the flow of execution of the services and the rules, the item set generator performs the job of generating the predictive incident table which aids in the assessment of risk. This is elucidated in the following sections.

The Audit log is a repository catalog for change history which stores the exact location along with the information about the domain, sub domain, business process and the service where the change occurred. It is similar to a log file which comprises of the time and date at which the change has occurred, the details about the owner who has the authority for that particular block of the business logic, the details regarding the changes and the information about the business process in which the change took place. The entire changes which have occurred and exceptions which have been thrown are recorded in audit log for incidence matching in the future. Emergency checks are also made to know whether the change is a critical one or not. This is performed by the complexity analyzer by comparing the current request with the previous incidents. The error handler handles the errors and exceptions thrown during the change process. The calibration engine extracts the results of the change evaluation and on taking the evaluated tuples as input to the cellular automata and probabilistic cellular automata, impact analysis and hence decision making is facilitated.

3.1. Methodology for Automated Verification of Composition Patterns

The automated verification of composition patterns which involves checking if a particular composition pattern is a part or subset of the given entire composition pattern. The methodology proposed in this study ensures this verification by scanning the service composition pattern and matching it with the entire composition pattern to identify if it is a part of the latter. The verification can be performed for service composition and operation composition. Consider the following LCS grammar where L is the business logic, S_1, S_2, S_3 are the services in composition, r_1, r_2, r_3, r_4 and r_5 are the rules in the composition:

$L \rightarrow S_1 \oplus S_2 \odot S_3$
$S_1 \rightarrow r_1 \oplus r_2$
$S_2 \rightarrow r_3$
$S_3 \rightarrow r_4 \oplus r_5 | \in$

For checking if a particular service or operation composition is a part of the given composition, the LCS grammar is scanned and the services or operations that begin (Pred) and immediately follow (Succ) each service or operation in some sentential form are identified. Considering the above grammar, the Pred and Succ functions are identified as follows:

Respect to service composition:

Pred (L) = Pred (S_1) =
{ r_1}
Pred (S_2) = { r_3}
Pred (S_3) = { r_4, ∈ }
Succ (L) = {$}
Succ (S_1) = Pred (S_2) + Succ (S_2) = { r_3, r_4, $}
Succ (S_2) = Pred (S_3) + Succ (S_3) = { r_4, $}
Succ (S_3) = {$}

With respect to service composition, the Pred (L) is ser-vice S_1 and Pred (S_1) in turn is r_1. The Pred (S_2) is identified to be r_3 and Pred (S_3) to be r_4. Likewise, the Succ (L) is identified to be $, Succ ($S_1$) to be r_3, r_4 and $, Succ ($S_2$) to be r_4 and $ and Succ (S_3) to be $. With Respect to Op-eration Composition:

Pred (L) = Pred	Succ (L) = {$}
(S_1) = { r_1} Pred	Succ (S_1) = {⊕}
(S_2) = {r_3} Pred	Succ (S_2) = {⊙}
(S_3) = {r_4, ∈ }	Succ (S_3) = {$}

With respect to operation composition, the Pred (L) is service S_1 and Pred (S_1) in turn is r_1. The Pred (S_2) is identified to be r_3 and Pred (S_3) to be r_4 and $. Likewise, the Succ (L) is identified to be $, Succ ($S_1$) to be ⊕, Succ ($S_2$) to be ⊙ and Succ ($S_3$) to be $.

3.1.1. Rule for Finding the Pred and Succ Function

If S is any symbol for representing the service in LCS grammar, let Pred (S) be the set of rules that begin the service derived from S. If $S^* \Rightarrow \in$, then ∈ is also in Pred (S). Define S_3(s), for the service S, to be the set of services S that can appear immediately following S_1 in some sentential form, i.e., the set of rules r such that there exists a derivation of the form $S^* \Rightarrow S_1 \oplus S_2$ for some services S_1 and S_2.

If S can be the last service in some extended form, then $ is in Succ (S). To compute Succ (S) for all services in S, apply the following rules until nothing can be added to Succ (S) set:

- Place $ in Succ (S), where S is the initial service composition to be invoked and $ indicates the end of service composition set
- If there is an LCS production of the form $S_1 \rightarrow S_1 \oplus S_2$ then Succ (S_2) = Succ (S_1)
- If there is an LCS production of the form $S_1 \rightarrow S_1 \odot S_2$, then Succ ($S_1$)⇒ Pred ($S_2$) + Succ ($S_2$)

3.2. Construction of Predictive Incident Table

After the Pred and Succ functions are identified, the predictive incident table is constructed. The algorithm for the construction of predictive incident **Table 1** is given below.

Table 1 and Fig. 2 shows the predictive incident table constructed based on the above algorithm. For each and every production in the LCS grammar, a valid entry is made into the predictive incident table. It is observed that Pred (L) is r_1. Therefore, the production $L \rightarrow S_1 \oplus S_2 \odot S_3$ has been added to M [L, r_1]. Pred (S_1) is r_1, Pred (S_2) is r_3, Pred (S_3) is r_4 and \$. Therefore the production $S_1 \rightarrow r_1 \oplus r_2$, $S_2 \rightarrow r_3$, $S_3 \rightarrow r_4 \oplus r_5$, $S_3 \rightarrow \in$ have been added to M [S_1, r_1], M [S_2, r_3], M [S_3, r_4] and M [S_3, \$] respectively. All other undefined entries of M are marked as error.

3.3. Composition Structure Validation

To predict the subset $S_1 \oplus S_2$ from the LCS, the following procedure is adopted. The procedure for the validation of composition structure L_2, involves finding the Pred (L_2) and Succ (L_2) and identifying the actual LCS set L_2 and attribute LCS set L_2. The Predictive Incident Table is then constructed and the composition structure validation is performed. The composition tree depicts the view of the services and operations in the form of the tree as shown in **Fig. 3 and 5**.

The algorithm for composition structure validation is given in **Fig. 4**. **Table 2** shows the composition structure validation for the LCS grammar discussed in the above sections.

To predict the subset $S_1 \oplus S_2$ from the LCS, the input stack is populated with the entries such that while an entry in M is not null and is a production of the form $S \rightarrow \alpha$, then is loaded on top of the stack instead of S. Input LCS set $S_1 \oplus S_2$ which is eventually $r_1 \oplus r_2 \odot r_3$, is loaded in the stack and the productions are traversed and the entries on the top of the input stack and input LCS set L_1 are compared. Whenever a match is found, the matching entries are popped from the stack. This process is continued till all the entries in the stack are popped and the service input symbol is null.

Then the service composition tree is constructed as shown in **Fig. 5**.

Thus LCS set L_1 is found to be a subset of the actual LCS set L_2.

3.3.1. Formulation of the Inference Metrics

The efficiency of the automated verification of composition patterns can be inferred from the following identified metrics.

Degree of automation is the measure which indicates the extent to which the composition patterns are automatically verified.

This is given by the following formulation:

$$DoAut = \sum_{i=1}^{n} \frac{Change\,\mathrm{Re}\,q_i}{n}$$

where, DoAut is the degree of automation, ChangeReqi is the change request i where the verification of composition pattern has been done automatically and n is the total number of change requests involving verification of composition patterns.

Table 1. Predictive incident table

δ (M)	r_1	r_2	r_3	$\oplus$	$\odot$	\$	r_4
L	$L \rightarrow S_1 \oplus S_2 \odot S_3$	E	E	E	E	E	E
S_1	$S_1 \rightarrow r_1 \oplus r_2$	E	E	E	E	E	E
S_2	E	E	$S_2 \rightarrow r_3$	E	E	E	E
S_3	E	E	E	E	E	$S_3 \rightarrow \epsilon$	$S_3 \rightarrow r_4 \oplus r_5$

Table 2. Composition structure validation

Input stack	Input LCS Set L_1	Predictive incident table
\$ L	$r_1 \oplus r_2 \odot r_3$ \$	$M(L, r_1) \Rightarrow [L \rightarrow S_1 \oplus S_2 \odot S_3]$
\$ $S_3 \odot S_2 \oplus S_1$	$r_1 \oplus r_2 \odot r_3$ \$	$M(S_1, r_1) \Rightarrow [S_1 \rightarrow r_1 \oplus r_2]$
\$ $S_3 \odot S_2 \oplus r_2 \oplus r_1$	$r_1 \oplus r_2 \odot r_3$ \$	Match
\$ $S_3 \odot S_2$	r_3 \$	$M(S_2, r_3) \Rightarrow [S_2 \rightarrow r_3]$
\$ $S_3 \odot r_3$	r_3 \$	Match
\$ S_3	\$	$M(S_3, \$) \Rightarrow [S_3 \rightarrow \epsilon]$
\$ ϵ	\$	Match-Accept

Algorithm Predictive_Incident_Table ()
Begin
Input: LCS grammar G.
Output: PIT table M.
Method:

1. For each production in S →r of the LCS grammar G, do Step 2 and 3.
2. For each rule r in Pred (S), add S →r to M [S,r].
3. If ∈ is in Pred (S), add S →r' to M [S, r'] for each rule r' in Succ (S). If ∈ is in Pred (S) and $ is in Succ (S), add S →∈ to M [S, $].
4. Make each undefined entry of M as error.

End

Fig. 2. Algorithm for Predictive Incident Table

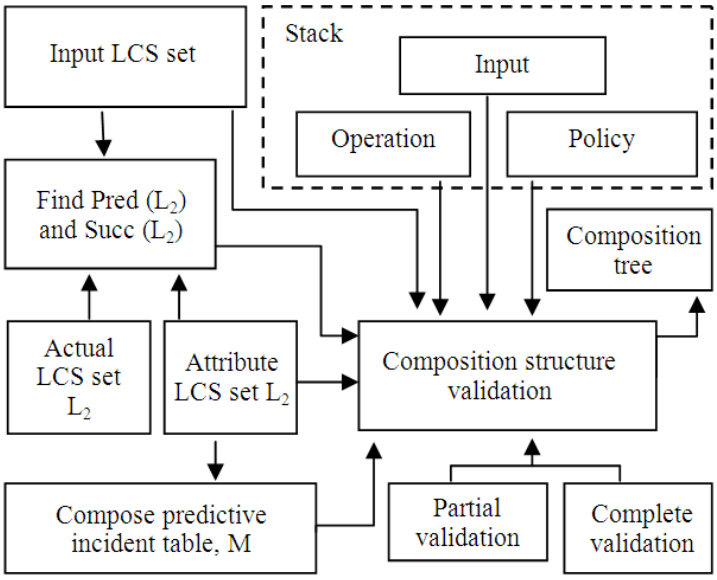

Fig. 3. Working of composition structure validation

Detection rate is the rate at which the patterns are successfully detected. It is defined as the ratio of number of change requests for which the composition patterns have been successfully detected to the total number of change requests. This is given by the following formulation:

$$Det\ Rate = \sum_{i=1}^{n} \frac{Succ\,\mathrm{Re}\,q_i}{n}$$

where, DetRate is the detection rate, SuccReqi is the change request i where the composition pattern has been successfully detected and n is the total number of change requests.

Error rate is the rate at which the patterns have been wrongly matched. It is given by the following formulation:

$$Err\ Rate = \sum_{i=1}^{n} \frac{Fail\,\mathrm{Re}\,q_i}{n}$$

where, ErrRate is the Error rate, FailReqi is the change request i where the composition pattern has been wrongly matched and n is the total number of change requests.

Algorithm Composition_Structure_Validation ( )

Begin

Input: LCS subset L_1 and actual LCS set L_2.

Output: Service Composition Tree (SCT)

Method: Set input pointer to point to the first service of input LCS set L_1 \$.

Repeat

1. If S be the state on top of the stack and r be the service symbol pointed to by input pointer, then find M [S, r].
2. If M [S, r] is not NULL and M [S, r] produces a production of the form S $\rightarrow \alpha$, then load α on the top of the stack in place of S.
3. If α be the service symbol on the top of the stack and β be the service symbol pointed by the input pointer, then

Begin

If $\alpha=\beta$ then

pop α from the top of the stack and move the input pointer so that it points to the next service symbol.

End

Until

The stack is empty and service input symbol is NULL.

Construct Service Composition Tree (SCT).

End

Fig. 4. Algorithm for composition structure validation

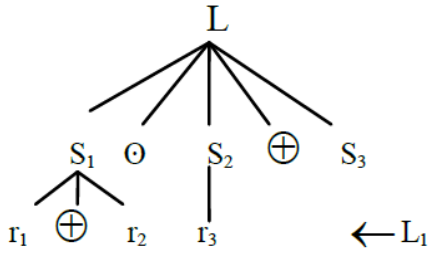

Fig. 5. Service composition tree

4. EXPERIMENTAL RESULTS

The experimental results for the automated verification of composition patterns have been elucidated in this section.

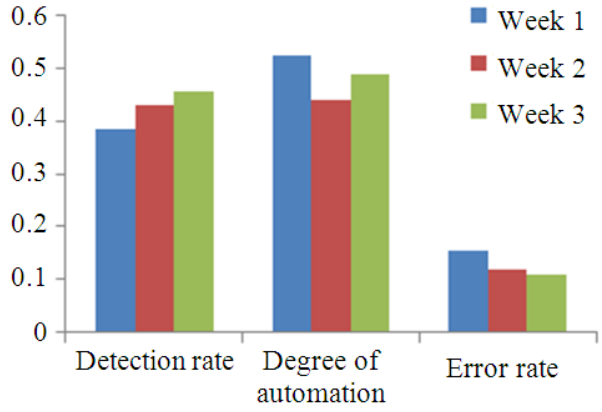

Fig. 6. Detection rate, degree of automation and error rate without automated verification of composition patterns

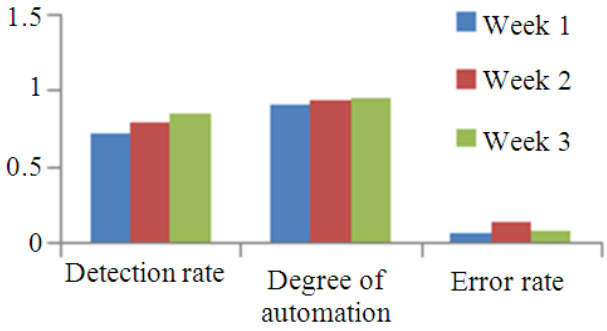

Fig. 7. Detection rate, degree of automation and error rate with automated verification of composition patterns

Table 3 Detection rate which is indicated by the no of change requests with successful matching results, Degree of automation which is indicated by the no of change requests handled automatically and the Error rate which is indicated by the no of change with wrongly matched results.

The graph in **Fig. 6** shows the detection rate, degree of automation and error rate observed for 3 weeks without automated verification of composition patterns.

The graph in **Fig. 7** shows the detection rate, degree of automation and error rate observed for 3 weeks with automated verification of composition patterns and depicts the significant improvement observed.

Table 3. Experimental results for automated verifica-tion of composition patterns

Dura-tion under observa-tion	No of arrived change requests demanding verification of composition patterns	No of change requests with suc-cessful matching results	No of change requests handled automatically	No of change re-quests with wrong matching re-sults
Week 1	290	212	267	21
Week 2	250	201	238	35
Week 3	263	226	251	23

5. DISCUSSION

Though there are existing works which have made considerable research in the area of change management and composition patterns like the change management framework proposed by (Liu *et al*., 2013) where managing changes in Long Term Composed Services (LCS) which deals with both functional and non functional change requests, they have not proposed any methodology for performing emergency changes and they have not found the various problems that will be caused due to the sudden changes. Architecture with a QoS based web service clustering method which helps us to select the best service that suits user quality preferences has been propounded by (Wu *et al*., 2009). But the service choice is done based on reputation and non functional aspects alone. Xiao and Urban (2012) have presented a recovery algorithm for service execution failure in the context of concurrent process execution. This recovery algorithm had been specifically designed to support a rule-based approach to user-defined correctness in execution environments that support a relaxed form of isolation for service execution (Xiao and Urban, 2012). The proposed system follows standard methodologies and ensures efficient composition pattern verification. The experimental results elucidated in the above section shows the detection rate, degree of automation and error rate of the composition pattern detection. This clearly indicates that the proposed system provides a dynamic and powerful platform for automatic verification of composition patterns and provides the scope for optimized incident matching. The **Fig. 6 and 7** indicate the experimental results observed. The detection rate, degree of automation and error rate are found to decrease when automatic verification of composition patterns is done. The graphs show the results obtained based on the observation made for a period of three weeks.

6. CONCLUSION

The need for an efficient change management approach and the increase in importunate change requests in long term composed services have thus necessitated the need for an automated way of verification of composition patterns. The paper has lime lighted the methodology behind the verification process and the impact of the prediction made. The experimental results prove the significance of the automation process and its role in efficient incident matching as when verification of composition patterns is done automatically and efficiently, the extent to which highly similar incidents are extracted is eventually enhanced and optimized. The proposal for verification of composition pattern alone is a limitation observed in this study as these pattern verifications are not only applicable for composition. The future work is to extend the proposed system into a unified dynamic model which would perform optimized incident matching not only for composed services but also for integrated services etc. Equipping the model with change management aspects by incorporating predictions based on risk involved can also be done.

7. REFERENCES

Akram, S., A. Bouguettaya, X. Liu, A. Haller and F. Rosenberg *et al*., 2010. A change management framework for service oriented enterprises. Int. J. Next-Generat. Comput., 1: 1-25.

David, K., H. Chua and A. Hossain, 2012. Predicting change propagation and impact on design schedule due to external changes. IEEE Trans. Eng. Manage., 59: 483-493. DOI: 10.1109/TEM.2011.2164082

Liu, X., A. Bouguettaya, J. Wu and L. Zhou, 2013. Ev-LCS: A system for the evolution of long-term composed services. IEEE Trans. Services Comput., 6: 102-115. DOI: 10.1109/TSC.2011.40

Liu, X., A. Bouguettaya, Q. Yu and Z. Malik, 2011a. Efficient change management in long-term composed services. Springer J. Service Oriented Comput. Applic., 5: 87-103. DOI: 10.1007/s11761-010-0074-3

Liu, X., S. Akram and A. Bouguettaya, 2011b. Change Management for Semantic Web Services. 1st Edn., Springer, New York, ISBN-10: 1441993290, pp: 182.

Plebani, P. and B. Pernici, 2009. URBE: Web service retrieval based on similarity evaluation. IEEE Trans. Knowledge Data Eng., 21: 1629-1642. DOI: 10.1109/TKDE.2009.35

Todd, L., A. Graves, F. Karr, J.S. Marron and H. Siy, 2000. Predicting fault incidence using software change history. IEEE Trans. Software Eng., 26: 653-661. DOI: 10.1109/32.859533

Wu, Y., J. Wu, J.J. Wang and S. Zhang, 2009. A multi-agent assistant dynamic resource management model of composite service. Proceedings of the IEEE Conference on Computational Intelligence and Software Engineering, Dec. 11-13, IEEE Xplore Press, Wuhan, pp: 01-04. DOI: 10.1109/CISE.2009.5364507

Xiao, Y. and S. Urban, 2012. Using rules and data dependencies for the recovery of concurrent processes in a service-oriented environment. IEEE Trans. Services Comput., 1: 59-71. DOI: 10.1109/TSC.2011.25