An Architecture for Self-Reconfiguration of Convergent Telecom Processes

Armando Ordóñez,
Jesus David Ramirez
University Foundation of Popayán
Popayán, Colombia
{jaordonez,jdramirez}@unicauca.edu.co

Paolo Falcarin
University of East London
London, UK
falcarin@uel.ac.uk

Oscar Mauricio Caicedo,
Lisandro Zambenedetti Granville
Universidad Rio Grande do Sul
Porto Alegre, Brazil
{omcrendon,granville}@inf.ufrgs.br

***Abstract*— A convergent process is usually defined as a composition of telecommunication and web services. Automated composition of convergent processes has been addressed actively in the last years. However, during execution phases some services may fail and therefore some mechanisms must be implemented for recovering automatically the normal execution. Furthermore, in Telecommunication environments, this process may be time-consuming and may violate the initial constraints established by the user's context and preferences. Our approach focuses in reducing the reconfiguration time while holding the initial constraints. To achieve this goal, this paper presents an iterative algorithm which does not replace individual services but whole regions of services, specified with Hierarchical Tasks Networks (HTNs). This algorithm is part of the reconfiguration module of the AUTO framework, whose architecture and performance are discussed to show that our approach can efficiently repair convergent processes in telecom environments.**

***Index Terms*— Automated reconfiguration, Automated planning, Convergent services, Service Composition.**

I. Introduction and Background

Convergent process may be defined as a structured set of services (telecommunication and Web services) that works in a coordinated manner to achieve a common goal [1]. One example of convergent process is a service that manages environmental early warnings (see Fig. 1). Environmental manager is in charge of decision making about environmental alarms and crops. In order to do so, the manager has information from sensor networks. The manager can also use Telecommunication and Web services to process basic data and send information to both farmers and sensors. Some typical requirements of such systems are: i) to calculate hydrological balance of the zone and receive the resulting map to the mobile, and ii) to emit an alarm to every farmer within a radius of 2 miles from the river if the river flow is greater than 15% of average". For the first request, the system must gather information from sensors, then the system uses hydrological services from the Internet, sending sensor data and maps fetched from Google maps. Finally the resulting image is sent via MMS to the user's mobile device. In the second request, sensor data are evaluated. If necessary, an emergency map is generated. This map is created drawing a radius of 2 miles from the sensor. To do so, the system uses geographic services and maps from internet. Finally, the system informs about the alarm to farmers inside the emergency area; the best way to send the information is selected: SMS, Cell Phone call, fixed telephone call, voice message. In both cases, services from Web and Telecommunications are used: these services work together and in coordination to save lives or help to make decisions about crops.

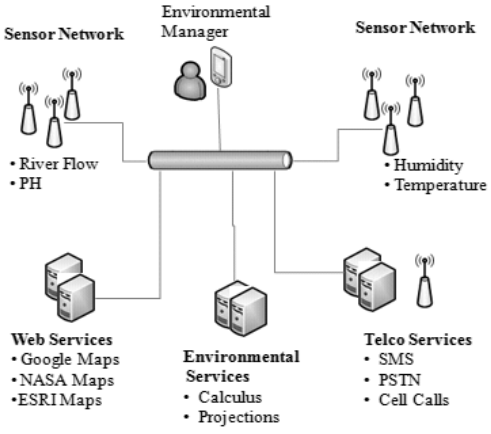

Fig. 1. Telecommunication and Web services interaction in Environmental Management Systems

Due to the dynamic nature of the Internet, Web Services may change, become unavailable and grow in number to unmanageable size. This means that manual synthesis and reconfiguration of convergent processes are unfeasible in practice and consequently automation of such tasks becomes necessary [2].

Identification and definition of guidelines for manual composition of convergent services are problems that have been investigated for years [18]; more recently, automation of different phases of convergent composition has been actively researched. Specifically, we have developed a framework called AUTO [3][4][5], which aims at supporting the automated composition of convergent services using automated planning for service composition and natural language processing for user request processing.

Among phases for automated composition, reconfiguration has been identified as one of the leading challenges in Service

2156-793X/15 $31.00 © 2015 IEEE
DOI 10.1109/PESOS.2015.9

oriented architectures [6][16][21]. Particularly, in the Telecom industry, high reliability is a crucial factor and the reconfiguration process must be as much transparent as possible. However, due to the fact that reconfiguration is time-consuming (the optimal service selection problem is NP-hard) [7], the number of halts and reconfigurations of the whole process must be minimized.

In previous works service reconfiguration mostly dealt with replacement of failing services, and consequently, most of the existing related approaches are focused on service selection of potential replacement services [9][10]. However, existing proposals are focused exclusively on Web Services, ignoring typical non-functional features of Telecom services, such as service deployment, real-time requirements, and event-based interactions. Sometimes replacing a failing service with one with different non-functional features cannot maintain initial user constraints. One interesting approach is the CHOReOS project (FP7-ICT-2009-5) "Large Scale Choreographies for the Future Internet (IP)" that aims for elaborating new methods and tools related to Future Internet ultra-large-scale (ULS) [20]. CHOReOS includes the study mechanism for adaptability and QoS-awareness based in monitoring choreography-based service composition in service-oriented systems. CHOReOS present a general approach applicable to different scenarios; however the authors did not consider the Telecommunication domain.

Initial user constraints are a key issue in telecommunications, as they may have been specified in a service level agreement. For example, some users or applications may require a precise maximum response time whereas others may be more concerned about low service costs. These user constraints must be considered during reconfiguration as well as they were considered during the initial automated composition phase.

Previously we have presented an approach for self-reconfiguration of convergent telecom processes based on automated planning [4]. In our previous approach, if a failure appeared in a service, this faulting service and the subsequent services in the convergent processes were replaced by another set of services. Besides, user preferences were considered during automated composition of the original plan, but the updated preferences (user situation and context may change) were not considered during reconfiguration. This approach presented some drawbacks: in telecommunication domain, it is not desirable to change the entire process but only the failing services [6]; besides the new services must be deployed into executable telecom environments, which is usually a time-consuming task. Additionally, when a new process is created from scratch, the time required for undoing the actions performed before the failure, must be taken into account.

In this paper we present an algorithm for automated reconfiguration replacing only the faulty region; the faulty region may be defined as a set of services that includes the service that caused the failure and their adjacent services. Two services are adjacent if one depends on the other, this dependency is established at design time.

Our approach can possibly reduce the overhead of repairing a complete faulty service process, since fewer services are involved in reconfiguration. Furthermore our approach decreases the time for undoing the task and recreating the whole convergent process.

Other projects for reconfiguration have also been reported in literature [7][8][17]. However, these works concentrate on Web service selection of replacements, and the initial requirements of users are not considered. Our work shares the perspective of Lin et al. [6] which propose a region-based service reconfiguration that maintains the Quality of Service (QoS) initially defined by user requirements. The authors presented a mechanism for selecting a replacement for a region surrounding the failing services. Unlike Lin et al., our approach is based on automated planning with preferences for performing the replacement of failing regions; besides, Lin's approach is focused exclusively in Web services, while our approach includes Telecom services.

In this context, the main contributions of this work are: an architecture for implementing automated reconfiguration in production telecom environments, an algorithm for region based reconfiguration supported in automated planning, and a performance report of the reconfiguration in AUTO.

The rest of this paper is organized as follows: section 2 depicts the system architecture and the main components of AUTO [4]; section 3 presents the reconfiguration architecture and algorithm for reconfiguration of convergent processes, Section 4 depicts the prototype as well as the performance evaluation, and finally, section 5 draws the conclusions and discusses future work.

II. OVERVIEW OF AUTO ARCHITECTURE

This section briefly depicts the components of the architecture for automated convergent composition AUTO (see figure 2). We refer the reader to [4] for a deeper explanation.

AUTO defines a series of sequential phases for automated service composition: *creation*, *synthesis*, *execution and reconfiguration*. The input method can be either by voice or text, which means that AUTO can be accessed from a broad range of devices.

The *Creation phase* decomposes the request in constitutive parts: the Natural Language Analysis deals with the treatment of the user request expressed in natural language. In AUTO, the transformation from natural language to planning language is performed by a module composed of a set of underlying components. The functional aspects and implementation of this module has been already described elsewhere [4], so we will only provide here a high-level description. The processing of the request in natural language is a set of sequential steps. First, the input is split into tokens, generating simple lexical units from complex sentences. Next, individual units are filtered and tagged according to their grammatical category. Additionally, each token is labeled as either "Control", "Functional" or "Situational" and classified according to three dimensions: device, user and situation. Next, information gathered from User Profile using the user's ID is added to the request. Finally,

the request is translated into a planning instance from which a service composition is computed.

These requests as well the context information (user preferences, device capabilities and situational context) are translated into a problem file expressed in Planning Domain Definition Language (PDDL). The automatically generated problem in planning language is the input sent to the *Synthesis phase*, based on the *Planning and LEarning Architecture* (PELEA) [11], which performs the synthesis of a plan representing the convergent process using automated planning.

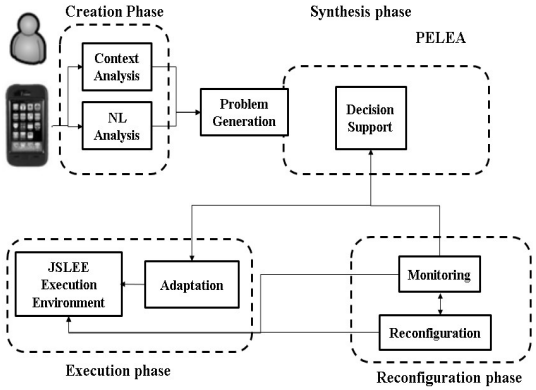

Fig. 2. Architecture of AUTO

AUTO uses a robust execution environment for telecommunications applications called Java Service Logic Execution Environment (JSLEE). The integration between PELEA and JSLEE is done in the *Adaptation module*. The adaptation module takes the synthesized plans and creates an executable convergent process. To do so, the adaptation module associates the planning operations to Java Snippets and generates a composite service out of JSLEE Service Building Blocks (SBB). While in other JSLEE solutions [18], web services are invoked through a SOAP resource adaptor, in our solution instead, SBBs are the basic components of the JSLEE architecture and are responsible of invoking Web services and Telecom functionalities. In situations in which the status and characteristics of the services may change, AUTO can monitor the execution of the composite services and also repair the current plan if needed. The reconfiguration process will be explained in detail in the next section

III. AUTOMATED RECONFIGURATION

The execution flow of AUTO tasks within a reconfiguration process is described in figure 3, and it is made of 10 phases (see fig. 3):

The *Synthesis* phase creates the *abstract plan* that represents the convergent process ((1) in fig. 3). However, telecom requirements require the user to be served immediately, so the elapsed time spent building the plan must be minimized. Nevertheless, in order to get the plan that represents the best solution, a big computational effort is necessary. On the average case finding the optimal solution requires exploring a very significant part of the search space, which makes such an effort impractical. Therefore, the best solution in a given time-window may be acceptable to begin the execution, and afterwards the planner may refine the plan while the first plan is executed".

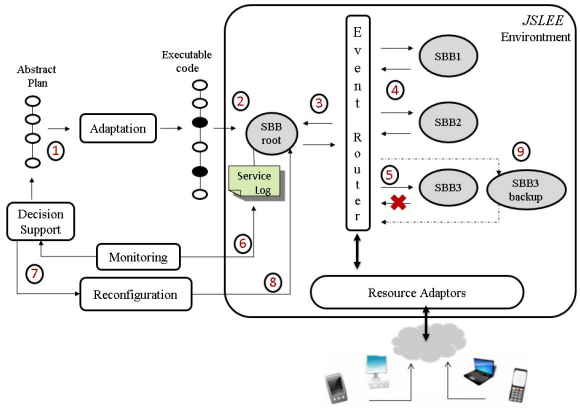

Fig. 3. Reconfiguration schema in AUTO

The additional plans generated after the initial solution comprise the *ranking of alternative plans* that are possibly used later. Later, the *Adaptation* module creates an executable convergent process in *JSLEE* through a *SBB root* (2). *JSLEE* is based on events, so during execution the *SBB root* controls the execution of individual SBB (*SBB1, SBB2* and *SBB3* in Figure 3) which represent individual services. The communication between SBBs is transmitted through the *Event Router*; similarly the communication between SBB and external networks is transferred by the *Resource adaptors*.

During the normal execution, *SBB root* invokes *SBB1* and *SBB2* using events (3), and get the successful response event (4). However, it may happen that *SBB3* does not receive the response event or it may generate an error (5). *SBB root* tracks all the execution results as well as the new "*state of the world*" in the server log (6). The *state of the world* is the set of information described in *planning language* that indicates the preconditions and post-conditions associated with execution of each service. For instance, a payment service changes the state of the world from "(not-paid)" to "(paid)", the next section describes in detail the services representation.

The *Monitoring module* gathers information from the execution of the convergent process execution in order to collect necessary data that will be necessary for determining if an error is present. This module gets information from the server log and invokes the decision support module for determining if *SBB3* presents an error. In order to identify if an error is present, the system compare the result of the invocation of a Service with a set of acceptable values specified at design time.

In case of failure, the reconfiguration algorithm is invoked. This algorithm aims for replacing *SBB3* and some surrounded services by other service or set of services that performs the same functionality. The algorithm starts from the current state of the world (represented in *planning language*) and performs a search in the existing services trying to minimize the set of

services to be replaced. To do so, *Monitoring module* invokes the *Decision Support module* based on automated planning ((7) in Figure 3). If the calculated region is very large or if the error is presented in the first service, so other plan is selected from the *ranking of alternative plans*. Regardless if a region or a whole process is changed, the *Reconfiguration module* performs the changes in the Server using Javassist ((8) in Figure 3), this approach is described deeply in [12]. Finally the replaced services or set of services replacing the functionality of *SBB3* respond to the *SBB Root* and the execution continues ((9) in Figure 3).

A. Service Representation

In order to establish an association between *convergent processes* and SBB in JSLEE, the synthesized processes are translated into Java components. To do so, the abstract convergent processes are integrated with execution patterns (*conditional*, *fork, join,...)* defined as Java snippets. This process is performed at run time.

As explained before, *convergent process*es generated in Java are monitored in the *Monitoring module*. The monitoring in AUTO is done using JSLEE alarms. Alarms monitor the execution of the services, and save the information into the log. The source code of alarms is inserted in the Java code during the translation between abstract *convergent processes* into SBB Components in the *Adaptation module*.

Regarding the alarms, two issues must be considered: firstly, if the number of alarms added to the SBB is very high, the quality of monitoring will be reasonably good but executing the verification process of each service so frequently may negatively affect the performance. This means that, in order to avoid an excessive overhead during monitoring, the number of alarms must be minimized. Secondly, if the failure is raised in more than one service, it could be necessary to re-plan and/or re-adapt the entire process. Besides, it could be necessary to undo all the tasks performed until the error occurred. The latter may be very time consuming. To address these issues, AUTO incorporates the region-based reconfiguration that will be explained later.

B. Service Modeling

AUTO uses automated planning as the mechanism for synthesis and reconfiguration tasks. There are different approaches for automated planning. In our approach we use HTN planning. In HTN planning a compound *task* is decomposed into smaller tasks using a *method*. The low level *task* that cannot be decomposed further is called a planning *operator*. In AUTO the highest level *task* or *goal* is extracted from the user request and the lowest level planning *operator* represents a primitive service directly associated with implementation of Web or telecom services. The planning process synthesizes plans, plans which are composed of a set of ordered operators or actions. These operators represent abstractions of Web and Telecom services (see Figure 4). *Operators* are represented using *input*, *output*, *pre-* and *post-* conditions. Examples of such operators are: CALL, LOCATE, SEND_SMS, TRANSLATE... Several implemented services may be able to perform the functionality required by an operator. Consequently, implemented services may be classified by their functionalities into operators. Every service modeled as an operator shares similar functionalities. These functionalities are usually the aforementioned *input*, *output*, *pre-* and *post-* conditions (IOPE). However, they may have different non-functional properties. For example, the operator INFORM can be performed by making a Voice call, by sending an SMS or by sending an MMS, which involves different services (See Figure 5).

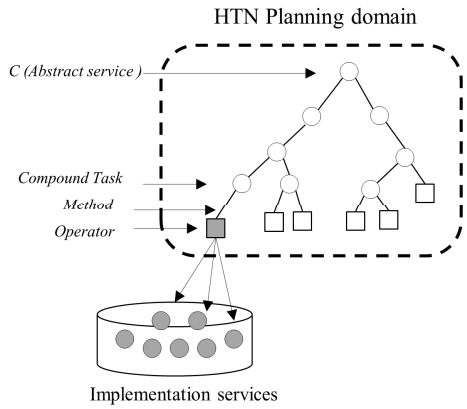

Fig. 4. Relation between operators and convergent services

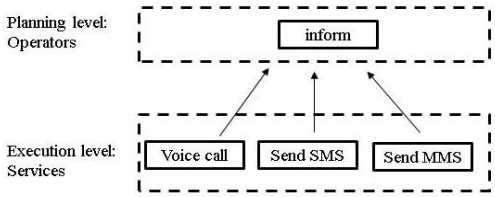

Fig. 5. Levels of services

In HTN a task must satisfy a set of preconditions and may produce some post conditions. This implies that a set of preconditions and post conditions can be fulfilled by a set of tasks or by a simple task. For example, to obtain the price of a product one may use a single service or a set of two services: one that gets the price and other that performs a currency conversion. Similarly, in the reconfiguration case, if any service malfunctions during the execution, the failing service can be replaced by one or several services that obtain the same effect. For example, an *operator* called “communicate” may be divided into functions to search the contact number (Search CN), locate contact (Locate C) and make a call (Call). Similarly, a set of tasks may be replaced by a new individual *operator* as long as the *pre-* and *post-* conditions correspond to the original service (see figure 6).For instance, in Figure 6 two paths, {*s1, s2, s3*} and {*s4, s5, s6, s7*}, may be traversed to achieve the goal, which means that the same high-level operator may be associated with different services. This is possible thanks to the fact that each operator may have several service candidates. For the sake of example, let us assume in

figure 6 that services *s5* and *s6* in the path 2 correspond to the functionality of the operator *o2* in the first graph.

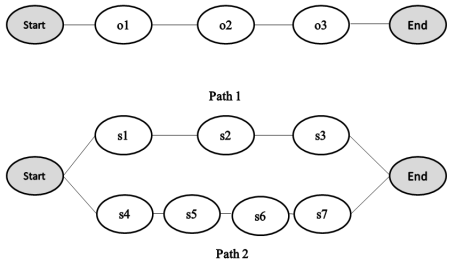

Fig. 6. Levels of services

C. Reconfiguration algorithm

The algorithm for service reconfiguration is shown below:

INPUT: faulty service si, threshold c
OUTPUT: replaced sub process Rf = {ri}
1: Set region ri = {si}; Rf = ∅, counter = 0
2: Counter = 0
3: WHILE Rf ≠ ∅ and counter < threshold c
4: counter = counter + 1
5: ri = {$r_{i\text{-}counter}$, ri, $r_{i+counter}$ }
6: Select a created plan ri' that meets IOPE and QoS of ri
7: IF ri' ≠ ∅
8: add ri' to Rf
9: END IF
10: IF Rf = ∅
11: HTN Planning for ri' that meets IOPE and QoS of ri
12: IF Rf ≠ ∅
13: add ri' to Rf
14: END IF
15: END IF
16: END WHILE
17: RETURN Rf

Let us suppose that a plan is generated and a service *si* presents a problem. Then, we must find a replacement for *si* in the implementation service repository. We must ensure that the new service has the same non-functional properties that respect the initial constraints of the plan, as defined by the user's preferences. If we cannot find a substitute service that meets the non-functional properties of *si*, we could try to replace the surrounding services, so we increase the counter and expand the region. To do so, we include the previous and the next service. Replacing a set of services together gives us more flexibility as long as the replacing services can meet the combined constraints needed. This way, the reconfiguration region can be extended or reduced in order to include services that comply with the aforementioned constraints. Anyway, note that, if the region includes many services for replacement, selecting alternative plans from the pre-computed ranking is probably a better option.

Given *p* faulty services and a reconfiguration threshold *c* (0 < *c* < 1) on the maximum number of services to be repaired, the algorithm first starts a loop (lines 3–17) to repeatedly expand the region. Inside the loop, the algorithm first tries to find a replacement region *ri'* for the failing service and adds it to the response *Rf* (line 6-9). If the existing plan does not exist, then the algorithm tries to recompose the region *ri'* using HTN planning (lines 10 -15) and adds it to the response *Rf*.

The central idea is to represent the region to repair using *input*, *output*, *pre-* and *post-* conditions (IOPE). This is the goal to reach using the planning algorithm that creates a new plan. Furthermore, the planning algorithm considers user preferences through the cost function that assigns a cost value to each generated plan [4]. The algorithm continues and, if the repaired region is very large and include too many nodes, the whole plan will be replaced by the existing ranking of plans.

IV. PROTOTYPE AND EXPERIMENTATION

The aim of this section is to verify the feasibility of our approach in real environments. This section describes three tests. The first test analyzes if the region-based reconfiguration has a better performance that the traditional reconfiguration. The second test is focused in measuring the performance of the reconfiguration algorithm in the Telecom server. The third test aims at analyzing the execution time when many services require the reconfiguration process

First, we evaluate the performance of the reconfiguration using the region based algorithm vs. the traditional reconfiguration. The traditional method reconfigures the whole plan from scratch, by undoing the tasks performed until reaching the task before the error occurred. If the error happened at the beginning, then there is no need for undo any task. Conversely, if the error showed up at the last node it is then necessary to re-design the entire plan or to select another plan from the ranking.

We test a plan with 10 nodes (services). Next, we assumed than an error occurred at different nodes of the path from 1st to 10th node (X axis in fig. 7). For representing undo tasks in our experiment, we performed a Web service invocation. As expected, time increases linearly with the number of tasks that must be undone; the higher the level of the node where the error is detected, the higher is the time spent in undoing the previous tasks (continuous line in figure 7). In figure 7 the dotted line represents the reconfiguration using the region-based algorithm; the region starts with a length of one node and expands. The region based reconfiguration based may find the solution at the first iteration of the algorithm (region equal to one node) or may find the solution at a posterior iteration. We established 5 nodes for this experiment, so the algorithm is performed recursively 3 times. It begins with a single node to be repaired and increasing the size of the region by two at each step (1, 3 and 5 nodes respectively). The average time spent by the region-based reconfiguration is lower than 100 milliseconds for the whole process (dotted line in fig. 7). The experiment was performed on an Intel Pentium Dual Core 2.2 GHz processor, equipped with 4 GB of RAM. The environment was executed using an instance of Eclipse Kepler service release 1

and JDK 1.7_01, and a local Web service for undoing the tasks. For the planning process, a HTN based planner was used [13].

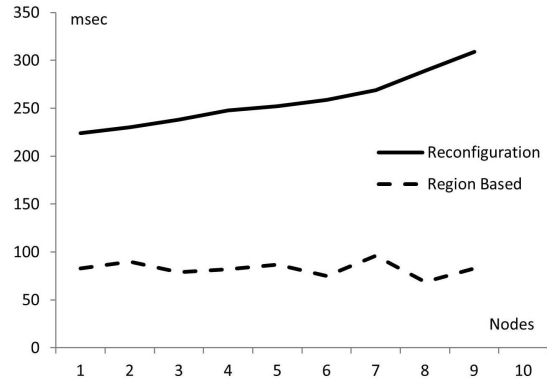

Fig. 7. Execution time of different invocations in Telecom environment

The second test is focused in measuring the performance of the reconfiguration algorithm in the Telecom server. First we measure the average execution time of each service of the convergent process presented previously in figure 2. For calculating the average execution time, 10 samples were carried out. In figure 8 the reconfiguration phase is the one which takes the most of the time, being the average value of 35 ms. The HTTP transaction phase responds the user request in 31.3 ms and finally the average time of the SBB invocation without using reconfiguration is 11.4 ms using 40 samples. From the data above it may be inferred that reconfiguration is efficient due to the fact that telecom services may accept a delay time of 150 ms.

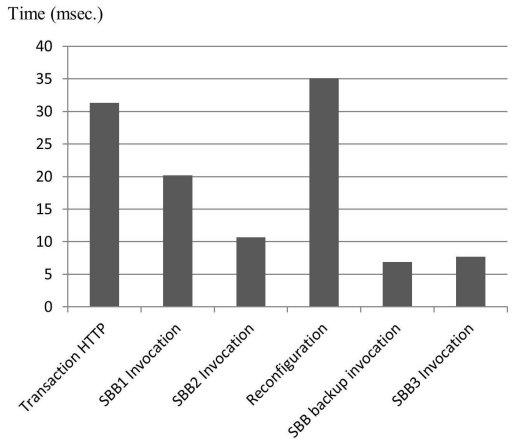

Fig. 8. Execution time of different invocations in Telecom environment

The third test aims at analyzing the execution time when many services require the reconfiguration process, i.e. the reconfiguration process is invoked at the same time for different services.

Figure 9 shows the execution time of the reconfiguration algorithm (Y axis) when the number of services invoking the module rises from 1 to 10 (X axis). Figure 9 shows that when reconfiguration is executed the processing time increases linearly; however in case of 10 services the processing time is close to 90 ms, thus the use of reconfiguration algorithm is feasible.

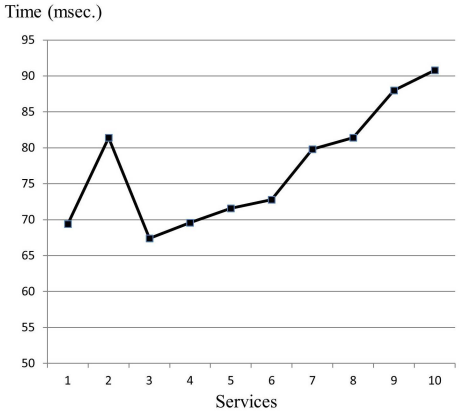

Fig. 9. Performance of reconfiguration with concurrent invocations

The second and third tests were measured in a PC with Core i5 Processor, 2.67 GHz and 4 GB of RAM, Debian 6.0.7. These values may be improved using a server with better hardware capabilities.

V. Conclusions

Convergent service composition requires that such services recover from failures efficiently. This work presents the results of our ongoing work towards the definition of a mechanism for planning-based reconfiguration of convergent services across the web and telecom domains. We described the approach for reconfiguring convergent processes and we presented an algorithm that allows reconfiguring only troublesome regions instead of re-planning the whole process or selecting another plan. Re-planning the whole process or selecting another plan from the ranking involves a big effort in undoing the actions of services executed before the failing service. Future work will include the use of different planning architectures for performing the reconfiguration of different failing regions at the same time, and we are also interested in performing further testing to evaluate performance and quality of the approach in cloud-based platforms for convergent services.

References

[1] Y. Cardinale and M. Rukoz, “Fault Tolerant Execution of Transactional Composite Web Services: An Approach,” Proc. Fifth Int, Conf. on Mobile Ubiquitous Computing, Systems, Services and Technologies, November 2011, pp. 158–164.

[2] Object Management Group. Profile for Advanced and Integrated Telecommunication Services (TelcoML), Object Management Group Standard. 2012.

[3] A. Ordonez, V. Alcazar, J.C. Corrales, P. Falcarin, “An Automated User-Centered Planning Framework for Decision Support in Environmental Early Warnings,” Proc. IBERAMIA, pp. 591-600, 2012.

[4] A. Ordonez, V. Alcázar, J.C. Corrales, and P. Falcarin, “Automated context aware composition of Advanced Telecom Services for environmental early warnings”. Expert Systems with Applications, vol. 41, no 13 2014

[5] A. Ordonez, J.C. Corrales, J. C. and P. Falcarin. , “HAUTO: Automated composition of convergent services based in HTN planning”. INGENIERÍA E INVESTIGACIÓN, Vol. 34, No. 1. 2014, pp.66-71,.

[6] K.-J. Lin, J. Zhang, Y. Zhai, and Xu, B., “The design and implementation of service process reconfiguration with end-to-end QoS constraints in SOA,” Service Oriented Computing and Applications, Vol. 4, No. 3, 2010, pp. 157–168.

[7] T. Yu, Y. Zhang and K. Lin, “Efficient algorithms for web services selection with end-to-end QoS constraints,” ACM Transactions on the Web (TWEB), Vol. 1, No. 1, 2007, pp. 6.

[8] D. Ardagna and B. Pernici, “Adaptive service composition in flexible processes,” IEEE Trans Softw Eng Vol. 33, No. 6, 2007, pp. 369–384.

[9] E. Kaldeli., A. Lazovik, and M. Aiello, M, “Continual Planning with Sensing for Web Service Composition,” AAAI, April 2011, pp. 1198–1203.

[10] K. Lin, J. Zhang Y. Zhai, “An efficient approach for service process reconfiguration in SOA with end-to-end QoS constraints,” In: Proc. of IEEE int. conf. on e-commerce technology (CEC). 2009.

[11] C. Guzman, V. Alcazar, D. Prior, E. Onainda, D. Borrajo, J. Fernandez-Olivares and E. Quintero, “PELEA: a domain-independent architecture for planning, execution and learning,” in Proc. Scheduling and Planning Applications workshop ICAPS conf., Freiburg, 2012, pp. 38–45.

[12] D. Adrada, E. Salazar, J. Rojas and J.C. Corrales, “Automatic Code Instrumentation for Converged Service Monitoring and Fault Detection,” Proc. Advanced Information Networking and Applications Workshops (WAINA), 2014 28th International Conference, May 2014, pp. 708-713

[13] E. Sirin, B. Parsia, D. Wu, J. Hendler, and D. Nau, “HTN planning for web service composition using SHOP2” Journal of Web Semantics, Vol. 1, No. 4, 2004, pp. 377–396.

[14] D.S. Nau, T.C. Au, O. Ilghami, U. Kuter, J.Murdock, D. Wu, and F. Yaman, “SHOP2: An HTN planning system”. J. Artif. Intell. Res, 2004, pp.379-404.

[15] C. Kun, J. Xu and S. Reiff-Marganiec. "Markov-HTN planning approach to enhance flexibility of automatic web service composition," Proc. IEEE Int. Conf. on Web Services, 2009, pp. 9-16.

[16] M. Shiaa, P. Falcarin; A. Pastor, F. Lecue; E. Silva, and L. Ferreira Pires, "Towards the Automation of the Service Composition Process: Case Study and Prototype Implementations", IEEE, ICT Mobile Summit, 2008.

[17] J. Wang, J. Yu, P. Falcarin, Y. Han; M. Morisio, "An Approach to Domain-Specific Reuse in Service-Oriented Environments", In Proc. of Int. Conf. on Software Reuse (ICSR-08), Springer, 2008.

[18] C. Venezia, P. Falcarin: “Communication Web Services Composition and Integration”, In Proc. of IEEE International Conference on Web Services (ICWS-06), 2006.

[19] C.A. Licciardi, P. Falcarin: “Service creation guidelines in Next Generation Networks”, in Proceedings of IEEE-ITU Int. Conf. on Intelligence in Networks (ICIN) 2003, pp. 139-144.

[20] M. Lescevica, E. Ginters, E. and R. Mazza, “Unified theory of acceptance and use of technology (UTAUT) for market analysis of FP7 CHOReOS products”. Procedia Computer Science, Vol. 26, 2013, pp.51-68.

[21] M. Belaunde, P. Falcarin, "Realizing an MDA and SOA Marriage for the Development of Mobile Services", In: 4th European Conference on Model Driven Architecture (ECMDA-2008), Springer, 2008, pp. 393-405

[22] J.P. Almeida, A. Baravaglio, M. Belaunde, P. Falcarin, "Service creation in the SPICE service platform". In Proceedings of the 17th Wireless World Research Forum Meeting (WWRF17), 2006.