Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 95 (2016) 353 – 360

Procedia Computer Science

Complex Adaptive Systems, Publication 6
Cihan H. Dagli, Editor in Chief
Conference Organized by Missouri University of Science and Technology
2016 - Los Angeles, CA

Comparing Drools and Ontology Reasoning Approaches for Automated Monitoring in Telecommunication Processes

Armando Ordóñez[a,]*, Luis Eraso[a], Hugo Ordóñez[b], Luis Merchan[b]

[a] *University Foundation of Popayán, 5St 8-58, Popayán, Colombia*
[b] *University of San Buenaventura, Av 10 de Mayo, Cali, Colombia*

Abstract

Automated reconfiguration is one of the crucial tasks in telecommunication service composition. The first step in reconfiguration is the monitoring phase. The problem of monitoring and error detection frequently appears in different telecommunications architectures. This article describes the main components of the architecture for monitoring module in AUTO framework. The monitoring approach is based on semantic technologies and ITIL framework. Equally, this paper presents an analysis and comparison of two approaches for the implementation of the module: Drools and semantic formalism. The results of this study may be applicable to other telecommunication domains.

Keywords: Service monitoring; automated reconfiguration; ontologies, rules, service composition.

1. Introduction and background

Telecommunication processes are composed of a 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. In this case, an environmental manager is in charge of decision making, as

* Corresponding author. Tel.: +0-000-000-0000 ; fax: +0-000-000-0000 .
E-mail address: jaordonez@unicauca.edu.co

doi:10.1016/j.procs.2016.09.345

well as define environmental alarms, and inform farmers and people in general about potential risks. In this way some processes can be defined in order to gather information from sensor networks, take decisions by means of web services, and inform farmers about possible risks using telecommunication services. A typical requirement of such systems is: "*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 this 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: short message service (SMS), cell phone call, fixed telephone call.

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 becomes necessary[2]. Recently, automation of different phases of convergent composition has been actively researched. Specifically, AUTO framework[3,4,5] aims at supporting automated composition of convergent services using automated planning and natural language processing.

Among phases for automated composition, reconfiguration has been identified as one of the leading challenges in Service oriented architectures[6]. Particularly, in the Telecom industry, high reliability is a crucial factor and the reconfiguration process must be as 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 this regard, a framework for automated reconfiguration in Telecom environments was presented[8,9]. This approach, proposes a region-based reconfiguration algorithm that maintains the Quality of Service (QoS) initially defined by user. The entire process doesn't change and automated planning is used for performing the replacement of failing regions. The first step to perform reconfiguration is error detection. This task may detect errors, classify them and select the action to be taken. Recently, different mechanisms were implemented for error detecting in Telecommunication servers[10,11]. The problem of detecting errors for reconfiguration frequently appears in different telecommunications architectures such as Internet of Things[12] and network management.

This paper describes an architecture for service monitoring supported in semantic technologies and ITIL model[13]. Equally, this paper compares two of the possible technologies for implementing this module according to different criteria. Results of this study offer important guidelines for designing reconfiguration architectures in other domains. The rest of this paper is organized as follows: section 2 depicts the main components of the reconfiguration framework as well as the algorithm for reconfiguration of convergent processes. Section 3 depicts the architecture of the monitoring module, section 4 presents the comparison between two approaches for implementing the monitoring module and finally, section 5 draws the conclusions and discusses future work.

2. Overview of reconfiguration framework

The reconfiguration framework is part of the architecture for automated convergent composition AUTO[4]. AUTO defines a series of sequential phases for automated service composition: *creation*, *synthesis*, *execution* and *reconfiguration*. The *creation phase* uses natural language processing for translating user requests and context information into a problem file expressed in automated planning language. The problem is the input sent to the *synthesis phase* which performs the synthesis of a plan representing the convergent process. For the execution AUTO uses a robust execution environment for telecommunications applications called Java Service Logic Execution Environment (JSLEE). In JSLEE services are represented as SBB (Service building blocks). AUTO can monitor execution of the composite services and also repair the current plan if needed.

Reconfiguration process in AUTO is made of 10 phases (see Fig. 1). The *synthesis* phase creates the *abstract plan* that represents the convergent process ((1) in Fig. 1). Next, 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 Fig. 1) 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 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

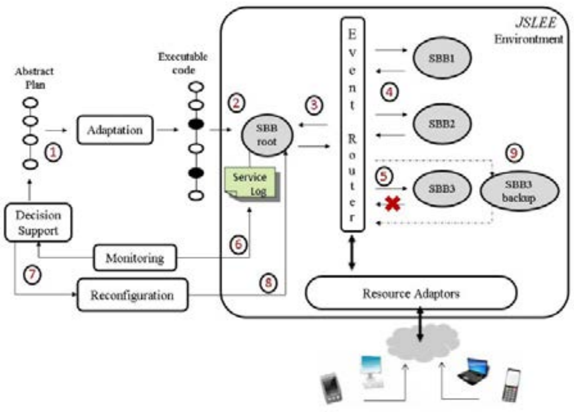

Fig. 1. Architecture of AUTO and monitoring module

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* (7) is on charge of gathering information from the execution of convergent processes in order to decide if an event is considered an error. This module gets information from the server log and invokes the *Decision support module* (7) for determining if SBB3 should be changed or modified. 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.As mentioned before, in order to make decisions about presence of errors some reasoning capabilities are required in the *monitoring module*. This is due to the fact that some services may produce results with wrong results, for example a service that retrieves the age of the user, may obtain an integer value of 250. This response may pass error detection controls at protocol and network level, but this response is nonetheless wrong (user age may be between 0 -120 in the domain). Next, the monitoring process is described deeply.

This module is based on events and analyzes the message exchange between services in order to detect inconsistent behavior[14]. This process require of two elements: i) the expected behavior of services may be represented in machine-readable format, and ii) an event handling mechanism to store, categorize and prioritize events may be defined. Regarding the first element, in order to define whether an event is classified as error or not, services must be modelled in machine-readable formats. That is, service behavior model must include both functional and non-functional requirements. Traditional technologies for services description such as Web Services Description Language (WSDL) define functional aspects like messages, types, and operations, meanwhile QoS requirements such as cost are defined using associated standards such as WS-Policy. Conversely, semantic approaches encourage the representation of services functionalities and QoS using Ontologies. AUTO uses ontologies for description of services as will be later detailed, specifically Service Oriented Architecture (SOA) Ontology[15]. Secondly, in order to classify events in errors or normal events, the *monitoring module* extends the events model described in the Information Technology Infrastructure Library (ITIL)[13] framework. ITIL defines activities and procedures to be executed when problems happen. ITIL categorizes events in: *informative* which indicate regular service operations, *warnings* indicating an unusual transaction and *exceptions* which may indicate failures. An *incident* is the occurrence of a warning or exception event. When an error occurs it can generate one or many incidents associated with service execution such: as a decrease in SLA (Service Agreements), sending of incorrect data, messages exceptions, or no response to certain requests. In this spirit, once an event has been categorized, it can be managed as an incident, a problem management or even a change. AUTO extends ITIL concepts by including new subtypes. For example *ServiceExecution* is a specialized type of information event whose purpose is to represent the execution of a service including: input and output value parameters, execution time, and response code information.

2.1. Architecture of monitoring module

Monitoring module interacts with the *Service log* and with the *Decision support* modules. The main components of the module are: *Service register, Service representation, Knowledge base, Working memory, Inference engine* and *Fault detector* (see Fig. 2). The *Decision support module* defines a set of services that will be monitored ((1) in Fig. 2). This list of monitored services is stored in the *Service Register*. The service register translates the service to its representation according to the SOA Ontology. Thus, every managed service has an associated representation in the *Knowledge Base* (2). Equally, associated constraints are stored in the Knowledge base using rules. Rules may be written using languages such as DRL (Drools rule language) or SWRL (Semantic Web Rules Language). Here, three types of rules had been defined: *Contract*, *Interface* and *Protocol* rules. Information about service execution is stored as a set of messages associated to events in the *Service log*. Thus monitoring module acts as a message handler, which includes tasks such as message interception of informative events and incidents. All these events are included as *facts* in the *working memory* (4) and verified by the *fault detector* in order to reason for possible Warnings or Exceptions.

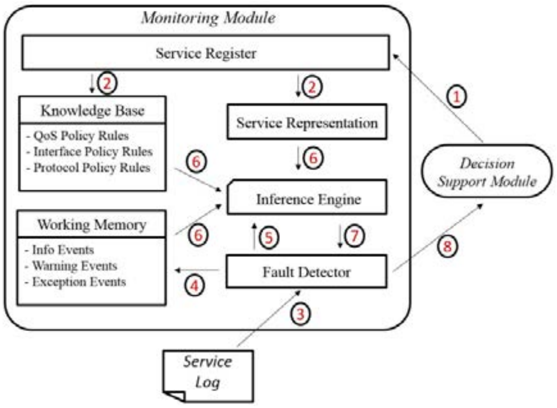

Fig. 2. Monitoring module in AUTO

In order to make decision, the *fault detector* uses information from: rules stored in the knowledge base, events stored in the working memory and service representation. If necessary, fault detector calls the *inference engine* (5). When a service is successfully executed, an *information event* is stored; this event includes timestamp, submitted values in inputs, obtained outputs, and execution time. When a service is not successful executed because inputs were not properly submitted (not compliant with defined service interface) so a *warning event* details wrong inputs. When a service does not respond properly despite having correct inputs (compliant to defined service interface) so an *Exception event* is generated. When *Exception events* are detected in the working memory this information is sent to decision *support module* (7) and (8) which is responsible for determining if it is necessary to change an individual service, a region or the entire process, thus ending the reconfiguration process.

2.2. Service levels for error detection

SOA ontology defines Service contract and Service interface as a fundamental part of service modelling. From this model, different levels have been defined for definition of rules that allow monitoring Telecommunications approaches. Rule-based representation provides flexibility and formality of services representation. Besides, due to its ease of use, Rules provide an efficient mechanism for definition, administration and modification of rules to respond to user requirements.

Level 1. Contract policy rules: The first level includes service interaction issues such as reliability or performance, as well as aspects such as cost or responsibilities associated with execution of services. An example of rules associated with this level may be*: "Service reliability is 99 %, i.e. in 100 executions only it may fail once"*

Level 2. Interface policy rules: At this level, ranges requirements, values types for inputs and outputs as well as restrictions of pre-conditions are formalized through policies. A concrete example may be a Health Service that uses a User ID as input and retrieves weight, body temperature, respiratory rate, heart rate, and blood pressure. Some example of restrictions are: "*Inputs must be integer between 0 and 999 (allowable values for User ID in the system)*"

Level 3. Protocol policy rules: During execution, services may return error messages associated with subjacent protocol; these messages may not be part service description. Consequently, it is necessary to include specific policies to determine the root cause of. A concrete example is the response messages of the HTTP protocol may be: "*If the result of the service includes Error code 404 (Not Found), so an exception is generated*".

3. Comparison of the two implementations

This section compares how the *Knowledge Base* and *Working memory* are implemented in two different technologies. On the one hand, Drools Engine, a Java based inference engine with its own rule language (DRL)[10]; and on the other hand, Web Ontology Language (OWL) as Service representation and SWRL as rule language[11]. Each approach includes mechanisms to represent services and implement the inference engine. Next, some conceptual differences between the two approaches are analyzed.

Regarding Service representation and working memory, both Drools and Semantic based approach use SOA Ontology as base to represent registered services. This given that SOA Ontology concepts are defined as platform independent, they can be used to represent both Web and Telco services. Non-functional requirements like legal aspects, interaction, performance, reliability could be defined as constraints and its effects related to Service Contract concept; in the same way functional requirements like input/output types and their restrictions are represented by means of service interface and information type concepts. OWL approach uses OWL-API[16] a high level API for working with OWL Ontologies closely aligned with the OWL 2 structural specification. In this approach SOA Ontology concepts may be represented directly as part of working memory. Therefore, each registered service may be mapped as an individual inside the ontology using concepts: Service, ServiceContract, ServiceInterface and InformationType. By its part, Drools platform uses instances of Java classes, so all concepts in the ontology were represented as plain classes. Consequently, the use of SOA Ontology directly is not possible and in order to implement the service register each new service may be created as an instance of the classes: Service, ServiceContract, ServiceInterface or InformationType and inserted later into the working memory. An advantage of OWL approach over Drools is the direct use of ontologies. Additionally, OWL is more expressive than object oriented approaches, therefore it will have a better representation and inference. When a service is registered into the inference engine, it becomes part of working memory immediately.

Regarding Knowledge base, this is composed of policy rules that describe constraints for monitoring in three different layers. After a formal representation of services is done, specific rules representing constraint must be defined. These constrains allow inferring occurrence of incidents based on incoming events, and categorizing incidents in Exceptions and Warnings. In Drools, rules are written in DRL language, an open source but non-standard language. In DRL syntax, every rule has a name that identifies it, equally, the "when" section defines restrictions to be accomplished for the rule to be triggered, and a "then" section defines actions to take when a rule is triggered. Fig. 3 shows an example of a rule implementing the policy that throws an exception; this exception is thrown if there is a violation of a restriction for maximum and minimum values in a service output. In the "when" section initially it is verified that there is a service ($sName) registered in working memory, additionally it verifies that there are call events to this service (ServiceCallEvent). Then it is verified that the service interface ($sInterface.outputs) contains numeric outputs, specifically the type INT or FLOAT. From the service call events (ServiceCallEvent) the output parameters values ($pValue) are taken. Finally the output values ($pValue) with the maximum permissible values according to restrictions are checked. If the rule condition is met, that is, if an output value is out of range, so the execution will be launched, i.e., the code in "then" section is executed. In the example a new exception event (NumberRangeValueExceptionEvent) is generated, next this event is added to the event log, thus the fault detector notices the error and send the event to the decision support module.

```
rule "ConstraintNumberRangeValue_PolicyRule" dialect "mvel"
when
   Service($sName:serviceName, $sInterface:serviceInterface )
   ServiceCallEvent(serviceName==$sName, $callEventOutputs:outputs )
   $infoItem: InfoType($itName:name, dataType in (DataType.INT, DataType.FLOAT))
        from $sInterface.outputs
   $pValue: ParameterValueBinding( parameter==$itName, $dataValue: dataValue )
        from $callEventOutputs
   $constraint : ConstraintNumberRangeValue( informationType == $infoItem,
                maxValue <= toNumber($dataValue, $infoItem) )
        from $sInterface.constraints
   $incidenceRegister : IncidenceRegister()
then
   NumberRangeValueExceptionEvent exception =
        new NumberRangeValueExceptionEvent( $sName, $constraint, $pValue );
   $incidenceRegister.eventList.add( exception )
end
```

Fig. 3. DRL rules implementing constraint policies for maximum values

A similar constraint may be built using SWRL language. Fig. 4. It describes two rules associated with the constraint RangeValueConstraint. These rules describe the occurrence of an Exception when a parameter value is outside of allowed values. In the first rule, the conditions are: a ServiceExecution exists, it has a Parameter which corresponds to InformationType with a RangeValueConstraint, and the parameter value is greater than the allowed value (swrlb:greaterThan). If these conditions are met, so it is possible to infer that this parameter has an exception or in other words it is an individual of Exception class.

```
ServiceExecution(?x), hasParameter(?x, ?p), Parameter(?p), hasType(?p, ?t),
InformationType(?t), hasConstraint(?t, ?c), RangeValueConstraint(?c), hasValue(?p, ?v),
hasMaxValue(?c, ?maxvalue), swrlb:greaterThan(?v, ?maxvalue) -> Exception(?p)

ServiceExecution(?x), hasParameter(?x, ?p), Parameter(?p), hasType(?p, ?t),
InformationType(?t), hasConstraint(?t, ?c), RangeValueConstraint(?c), hasValue(?p, ?v),
hasMinValue(?c, ?minvalue), swrlb:lessThan(?v, ?minvalue) -> Exception(?p)
```

Fig. 4. SWRL rules implementing constraint policies for max and min values

As can be noted from the previous examples, SWRL rules are simpler and easier to define than DRL rules. This is because they follow a defined standard which includes predefined operation such as mathematical functions "greater than" (swrlb:greaterThan) and "less than" (swrlb:lessThan) as well as other functions to work with dates, Strings, URIs and Lists. On the other hand, similar features can be found in the Java language which may be imported into a DRL rule. However this may complicate the understanding of rules as in the case of the “toNumber” function inside the DRL rule described before in Fig. 3.

Regarding Inference engine, Drools has its own inference engine integrated into the API which is based on the Rete[16] and Phreak[17] algorithms, this engine cannot be modified. In case of OWL approach, different OWL reasoners can be used[18]. In the present study the Pellet Reasoner was used because of its support to OWL 2 and SWRL operations. Discussion about technical comparisons of different reasoners and algorithms is out of scope of this paper; however, the new section presents some tests to compare the performance of the two alternatives.

In terms of expressiveness, OWL is presented as a suitable solution. Firstly, the transparent representation of OWL concepts allows using ontologies like SOA Ontology and OWL-S. Secondly, rules defined in SWRL allow exploding the advantages of a standard language. In spite of these advantages, the number of tools for editing these rules is limited. Regarding scalability it should be considered the widespread use of Drools in business environments. Therefore it has tools for managing distributed rules, tools to create Domain Specific Languages

(DSL) and create rules based on these languages, and APIs that provide interfaces with other components such as Enterprise Service Bus (ESB) and Business Process Management Engines (BPM).

4. Experimentation

The aim of the experimentation is to evaluate the performance of the two approaches (Drools and Semantic based approach) for implementing the crucial tasks of the monitoring process. The evaluated criteria are: i) creation time of the knowledge base, ii) Time of service registration and iii) Execution time of the inference engine.

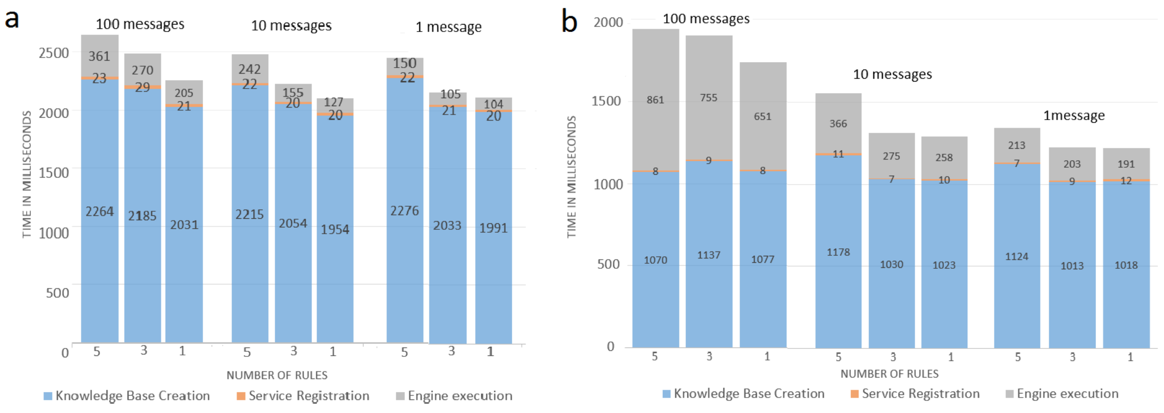

Fig. 5. (a) Execution time in Drools; (b) execution time in OWL

Tests were performed varying the number of rules in the knowledge base, initially 5 rules were added: 2 rules defined exceptions in output parameters, 2 rules defined warnings in the same parameters and 1 rule defined execution time as legal aspect. Equally, the execution of inference mechanism was performed with 100, 10 and 1 event message execution in order to detect incidents. Same tests were made but reducing the number of rules. First the warning rules were removed, so the test were performed with 3 rules. Second the execution time rule and one of exception in output parameters were removed, therefore the test was made just with one rule. The summary of evaluation is presented in Fig. 5.In both cases the most time consuming task is Knowledge base creation. In Drools the average time is 1892.5ms with a maximum of 2276ms when it uses 5 rules and 1991ms when it uses 1 rule, a difference of 285ms. In case of OWL the average time is 1074.4ms with a maximum of 1178 with 5 rules and 1013ms with 3 rules, a difference of 165ms. We can see that OWL approach is faster than Drools, but this task is executed just once when rules are defined. Therefore, the consuming time of other tasks should be evaluated. Service registration is the less consuming task with an average of 22ms in Drools approach and 9ms in OWL approach. There is no important difference in time consuming between 5 and 1 rules evaluation; but like before OWL is faster than Drools. This task is executed every time a new service is included for monitoring. The most significant task is engine execution because it is executed during server execution in order to find new incidences in service execution. The results evidence important differences when varying the number of messages and number of rules into the working memory. The average time in Drools approach is 191ms with a maximum of 361ms with 100 messages and 5 rules. In case of OWL the average is 419,2ms with a maximum time of 861ms whit 100 messages and 5 rules. Here we can see a significant increase in case of OWL when number of rules and messages rise.

5. Conclusions

Automated reconfiguration is one of the crucial tasks in telecommunications service composition. The first step in reconfiguration is the monitoring phase. The problem of monitoring and error detection frequently appears in different telecommunications architectures such as Internet of Things and network management. This article describes the main components of the architecture for monitoring module in AUTO framework. The monitoring approach is based on semantic technologies and ITIL framework. Equally, this paper presents an analysis and comparison of two approaches for the implementation of the module: Drools and Semantic based. The results may be applicable to other domains.

The results evidences that the OWL implementation offers advantages in terms of rules expressiveness but Drools is a good solution in large environments because of its scalability and performance. As future work more detailed experimentation will be performed including a higher number of registered services and validation of accuracy in error detection. Finally, once a big set of events and incidents have been gathered, it is planned to use data mining approaches in order to be able to predict errors based on previous executions.

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 on 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. A. Ordóñez, H. Ordóñez, C. Figueroa, C. Cobos, and J. C. Corrales "Dynamic reconfiguration of composite convergent services supported by multimodal search". In Business Information Systems 2015, pp. 127–139. Springer International Publishing.
9. Armando Ordonez, Paolo Falcarin, Lisandro Zambenedetti. An Architecture for Self-reconfiguration of Convergent Telecom Processes Proceedings of the 7th PESOS Workshowp. The 37th International Conference on Software Engineering, Florence, Italy. 2015
10. A. Ordonez, L. Eraso, P. Falcarin "Rule-based monitoring and error detecting for converged telecommunication processes," in SAI Intelligent Systems Conference (IntelliSys), 2015 , vol., no., pp.705-713, 10-11 Nov. 2015.
11. L. Eraso, A. Rodríguez, A. Ordonez, O. Caicedo "Ontology Based Monitoring and Error Detection in Converged Telecommunication Processes," Euro American Conference on Telematics and Information Systems (EATIS), 2016, May. 2016.
12. Cirani, S., Davoli, L., Ferrari, G., Léone, R., Medagliani, P., Picone, M., & Veltri, L. (2014). A scalable and self-configuring architecture for service discovery in the internet of things. Internet of Things Journal, IEEE, 1(5), 508-521.
13. F. Cervone, "ITIL: a framework for managing digital library services," OCLC Systems & Services: International digital library perspectives, vol. 24, no. 2, pp. 87–90, 2008.
14. K. Bratanis, D. Dranidis, and A. J. H. Simons, "An extensible architecture for run-time monitoring of conversational web services," in Proceedings of the 3rd International Workshop on Monitoring, Adaptation and Beyond, New York, NY, USA, 2010, pp. 9–16.
15. The Open Group, "Technical Standard: Service Oriented Architecture Ontology." Berkshire: The Open Group, 2010. 1-931624-88-7.
16. C. L. Forgy, "Rete: A fast algorithm for the many pattern/many object pattern match problem," Artificial intelligence, vol. 19, no. 1, pp. 17–37, 1982.
17. Mark Proctor, "R.I.P. RETE time to get PHREAKY," Drools & jBPM, 01-Nov-2013. [Online]. Available: http://blog.athico.com/2013/11/rip-rete-time-to-get-phreaky.html. [Accessed: 01-Jul-2016]
18. M. Horridge and S. Bechhofer, "The owl api: A java api for owl ontologies," Semantic Web, vol. 2, no. 1, pp. 11–21, 2011.