DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Relatório Técnico

RT-MAC-2013-02

An Adaptive Enactment Engine for Complex Service Compositions on the Cloud

Leonardo Leite, Nelson Lago, Thiago Furtado,
Carlos Eduardo Moreira,
Daniel Cordeiro, Daniel Batista, Marco
Aurélio Gerosa, and Fabio Kon

Fevereiro de 2013

An Adaptive Enactment Engine for Complex Service Compositions on the Cloud

-

Technical Report # RT-MAC-2013-02

Leonardo Leite, Nelson Lago, Thiago Furtado, Carlos Eduardo Moreira, Daniel Cordeiro, Daniel Batista, Marco Aurélio Gerosa, and Fabio Kon

5 de julho de 2013

Department of Computer Science
Institute of Mathematics and Statistics
University of São Paulo

You may cite this document as:
Leonardo Leite, Nelson Lago, Thiago Furtado, Carlos Eduardo Moreira, Daniel Cordeiro, Daniel Batista, Marco Aurelio Gerosa, and Fabio Kon. An Adaptive Enactment Engine for Complex Service Compositions on the Cloud. Technical Report # RT-MAC-2013-02, Department of Computer Science, University of Sao Paulo, july, 2013.

Abstract. Software developers have today the unprecedented ability to combine different computational resources geographically spread around the globe into a powerful large-scale distributed computing platform. Service-oriented computing is an interesting approach for such platforms since it eases the development of very large-scale distributed systems, enabling the development of cross-organizational business processes based on the composition of multiple web services. Enabling the deployment of such large-scale web service compositions on a very large number of computational resources requires specific mechanisms for automation, reliability, and adaptability.
In this paper, we present the CHOReOS Enactment Engine, a novel, extensible, open source middleware system that provides a platform for the automation of distributed deployment of complex web service compositions on hybrid cloud computing environments. The Enactment Engine also supports auto-scaling and autonomic cloud resource management. We show how the Enactment Engine enables the deployment of service compositions in an automated, scalable and reproducible way, even in the presence of failures in the cloud infrastructure. Our experiments show that the Enactment Engine scales well when there is an increase in the number of services to be deployed and it is capable of providing to applications a large amount of cloud resources efficiently.

Keywords: cloud computing, adaptive systems, web service compositions, automated deployment

1 Introduction

Services are autonomous, platform-independent entities that can be described, published, discovered, and loosely coupled in novel ways to realize sophisticated business processes [1]. The composition model where a centralized coordinator manages all the interactions among the services is called *orchestration* (the coordinator is called orchestrator) [2]. On the other hand, *choreography* [3] is the composition model where the knowledge about the control flow is distributed among the participants, i.e., each service acts autonomously and knows when to execute its operations and with whom to interact.

The current trend in the evolution of the Internet is leading us to a scenario in which systems will be composed of a large number of distributed services running on heterogeneous platforms hosted on a variety of mobile and cloud-based infrastructures. To illustrate such an environment, let us imagine a scenario in which a Future Airport is completely automated with the help of sophisticated service compositions.

First, in such a complex and large-scale scenario, it would not be reasonable to have a centralized point of control, thus a choreography model would be desirable. These choreographies would implement distributed business processes with

multiple participating organizations (airport authority, airlines, travel agencies, etc.). Many of the services provided within the airport could be automated according to the passengers personal information, specially their flight information. Examples of possible automations suggested by companies with expertise in the airport domain [4] include: (1) the parking service may indicate the parking lot which is the nearest to the boarding gate (2) catering services may prioritize meals to passengers with earlier flight times, (3) airline companies may schedule automatically hotel and taxi services to passengers when a flight is canceled, and (4) all airlines may be informed at once by Air Traffic Control about flight cancellations due to bad weather at destinations.

This airport scenario turns into a large-scale choreography if we consider, for example, the actual numbers involved in airports, such as Heathrow[1] in London, which deals with more than 80 airlines, 190,000 passengers per day (peaks of 230,000), 6,000 employees, 1,000 take-offs and landings per day and 40 catering services. As all these businesses are automated and the devices carried by these people run services that are integrated in a very large service choreography, we will be approaching the landscape depicted in vision of the *Future Internet*, where millions of resources, people, and things will be online all the time establishing *ad-hoc* connections to compose several complex service compositions [5]. The current understanding of the industrial and research community is that the current Internet is not well equipped to address the Future Internet requirements [6]. Thus, it is necessary to devise mechanisms that work, scale, and can efficiently implement collaborations within large-scale distributed systems [7] like, for example, large-scale, complex, service choreographies.

The deployment process of large-scale choreographies is the particular challenge of the Future Internet this paper addresses. The large number of services and the distribution of them among several nodes increase the difficulties present in any deployment process, which usually is a time-consuming and error-prone activity [8]. To avoid such difficulties, researchers and practitioners advocate the use of automated deployment processes [8, 9].

The current trend to deploy large-scale systems is the usage of scalable resources provided by cloud computing services. Cloud computing enables automated provision of resources retrieved from shared pools [10]. The use of virtualized resources provided by the cloud service leverages the automation of the deployment process [9]. The Infrastructure as a Service (IaaS) model, e.g., Amazon EC2[2], provides programmatic access to virtualized resources such as virtual machines [11]. The creation of new application-dedicated virtual machines is what enables the deployment process to be automated and reproducible. On the other hand, in the Platform as a Service (PaaS) model, e.g., Google App Engine[3], application developers do not need to handle the virtualized environment, but only the application itself. Main features of PaaS providers are the automated

[1] http://www.heathrowairport.com
[2] http://www.amazon.com/ec2
[3] http://cloud.google.com/appengine

deployment of applications and transparent auto-scaling of underlying resources according to the application demand [11].

In this paper, we present the CHOReOS Enactment Engine, a novel, extensible, open source middleware system that provides a PaaS service for the deployment of complex service compositions. It enables a fully automated deployment process, including the preparation of the target environment and the scaling of resources as needed. The automated provisioning of new environments is built on top of third-party IaaS infrastructures. The Enactment Engine uses the IaaS service to support the deployment of choreographies, which can be provided in the Software as a Service (SaaS) model to end-users. This relationship between the cloud models and our architecture is depicted in Fig. 1. The Enactment Engine also provides features to enable services replication, which is used to adapt to dynamic fluctuations in the load. Finally, another useful feature is the automated deployment of middleware components on the cloud nodes to enable the easy execution of middleware services such as monitoring and ESB-based communication.

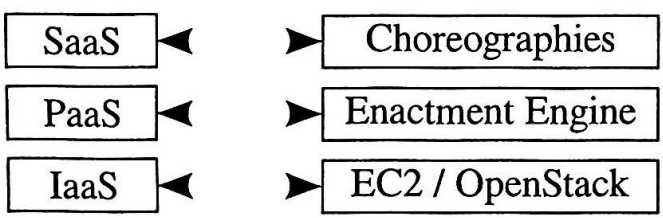

Fig. 1. Relationship between cloud computing models and our architecture

Unlike current PaaS solutions, the Enactment Engine is designed to support service compositions rather than web applications. Another difference is that the Enactment Engine is designed with an extensible architecture to overcome current limitations of PaaS solutions that restrict application developers' technological choices, such as IaaS provider and application programming language. Our solution also leverages past research results on deployment of distributed component-based systems, but adapting them to current environments by considering cloud-deployment and large-scale requirements. The most important cloud requirement is a flexible configuration model for services that takes into consideration the dynamic nature of the cloud [12], which results in the requirement that services should not know the exact address (URI) of their dependencies at design time. With regard to large-scale deployment, a central concern is related to properly handling failures – which are routine in Cloud environments – and unpredictable latencies in third-party components [13, 14]. In our case, the main third-party component with variant behavior is the IaaS provider, whose provisioning of virtual machines may take vastly different times for completion at each new request and even fail sometimes.

In this paper, we introduce the generic CHOReOS Enactment Engine architecture and detail its implementation, giving empirical evidences of its performance when deploying large-scale choreographies. In Section 2, we discuss related work on automated deployment of component-based systems and on current PaaS solutions. In Section 3, we explain the Enactment Engine architecture and its functionalities, while implementation details in a concrete context are showed in Section 4. In Section 5, we discuss the Enactment Engine performance and scalability when deploying large-scale choreographies. Finally, our conclusions are drawn in Section 6.

2 Related work

The current state of the art in distributed computational platforms enables the unprecedented ability to combine large amounts of computational resources (possibly geographically dispersed) into a powerful supercomputing platform. Grid and cloud computing platforms can now offer such ability to any developer in the world and are effectively being used to execute applications on a wide variety of domains.

It rapidly became clear that in order to support such variety of domains, the platform should provide full control over the provisioning and configuration of the computational nodes. The developer should be able not only to control the kind of hardware that will be provisioned for its application, but also to control the entire software stack: from the underlying operating system up to the application.

Early research aiming to automate the deployment process did not offered the possibility to customize the operating system. They were focused on the automated deployment, configuration, and load balancing of the components to be deployed on the available computing nodes. Usually conceived to manage a few hundreds of nodes, they basically work by taking applications described using a procedural [8] or a declarative [15, 16] language description.

The use of procedural language, i.e. scripts, is the most flexible method to configure and deploy applications. Software systems like TakTuk [17], Chef[4], Capistrano[5] or Nix [8] provide script languages that can be used to configure each node, deploy the application, and also test and build the applications on the nodes. Although flexible, the use of scripts is more error-prone. The scripts must be developed with the same rigor and accuracy used to develop the application itself.

TakTuk [17] is an open-source tool developed to execute interactive parallel tasks (such as cluster administration, or parallel application debugging and tuning) on a potentially large set of both homogeneous and heterogeneous remote machines. It is able to spread itself using an adaptive algorithm and sets up an interconnection network to transport commands and perform I/O multiplexing/demultiplexing. The TakTuk mechanics dynamically adapts itself to

[4] http://www.opscode.com/chef

[5] http://capistranorb.com

the environment (machine performance and current load, network contention) by using a reactive work-stealing algorithm that mixes local parallelization and work distribution.

Chef is a popular configuration management tool used by cloud computing practitioners. It provides a domain-specific language that facilitates the creation of the configuration scripts (also called "recipes" and "cookbooks", see more details at Section 3). It is implemented using a client/server architecture, where the information is propagated using a "pull" model: each node queries the server for the most recent configuration details, such as recipes, templates, and file distributions and performs most of the configuration work itself. The use of a pull model helps to keep the node configurations always up to date and can scale efficiently when used with a replicated NoSQL database.

The complexity of scripts written using procedural language systems like TakTuk and Chef can quickly increase when deploying large applications. The Enactment Engine was designed to deploy applications described as business processes. It uses a notation based on a declarative language (more details on Section 3). Declarative languages for automated deployment [15, 18] describe the application to be deployed in terms of its components and how they are structured. The idea is that a richer model of the application can help the deployment system to optimize the available platform by making a more informed choice of the nodes and allowing the optimization of the configuration process.

Quéma et al. [19] conducted an empirical study about the performance and scalability of the deployment process of component-based applications. They propose a decentralized, fault-tolerant mechanism that deploys the application using an hierarchical approach based on the application's architecture. This approach enabled an asynchronous and parallel execution of the components that defines the application. However, the hierarchical approach, as proposed by the authors, can only be applied to business processes whose communications can be modeled as trees.

One of the first large-scale platforms to support total control of the software stack (including the operating system) was Grid'5000 [20]. Its Kadeploy [20, 21] OS provisioning system has been designed not only to help system administrators install and manage clusters, but also to provide users with a flexible way to deploy their own operating systems (at runtime) on nodes for their experimentation needs. According to Jeanvoine et al. [21], the key features to achieve scalability when deploying and configuring thousands of computing nodes are: (i) the ability to execute parallel commands; (ii) an efficient file broadcast system; and (iii) windowed operations, to prevent resource intensive operations to be executed all at once (for example, rebooting a large number of nodes in the same room can generate electrical hazard). In Grid'5000, these features are achieved using Kadeploy in conjunction with TakTuk. The Enactment Engine achieves a similar result using the information about the application being deployed, the automatically generated Chef scripts and the underlying cloud computing IaaS platforms.

The Enactment Engine offers to developers access to hybrid cloud platforms using a PaaS model. Cloud computing offered as a PaaS eases the development of applications, by providing a set of tools and libraries that must be used by the developer. An application developed using this standardized model can be more easily deployed on the cloud computing platform. All the underlying details about the deployment, provisioning of resources, automatic load balancing, monitoring, etc. are responsibility of the platform provider. There are several cloud computing providers that offer access to their resources using a PaaS model, such as the Amazon Web Services (AWS) Elastic Beanstalk[6], the Google AppEngine, or the Microsoft Windows Azure[7].

A problem with applications developed to use the PaaS model is that the software becomes dependent on the tools and libraries provided by the chosen cloud computing provider. This dependency might lead to vendor lock-in and can become a problem if the developer first chooses the vendor and, afterwards, choose to use a different (or more than one) cloud computing provider. Two PaaS platforms offer services to support the execution of business processes: the Amazon Simple Workflow Service[8] and the Force.com Visual Process Manager[9]. Both platforms offer a simple coordination mechanism to control the order which each activity of a business process should be executed, i.e., an orchestrator.

The use of open source frameworks can help to mitigate the vendor lock-in problem. The Cloud Foundry[10] provides an open-source PaaS framework that can be deployed on a variety of cloud computing platforms (including AWS, VMware vSphere[11] and OpenStack[12]). The CompatibleOne [22] is an initiative to develop an energy-efficient open source cloud broker. The framework can manage any type of cloud services, which can be provided by heterogeneous service providers selected according to a Service Level Agreement (SLA).

3 Architecture

According to the OMG [23], the deployment process of component-based distributed applications consists of the following phases: *installation:* bringing software components packages under the deployer's control (e.g., by purchasing); *configuration:* editing configuration files of software components; *planning:* defining in which infrastructure node each component must run; *preparation:* moving components and middleware packages to the the target environment, so the software is ready to be executed; and *launch:* finally running the software. The first two phases are usually handled manually by the deployers, since they are less suited to be automated (for instance, setting security credentials). The

[6] http://aws.amazon.com/elasticbeanstalk
[7] http://www.windowsazure.com
[8] http://aws.amazon.com/swf
[9] http://www.salesforce.com/platform/process
[10] http://www.cloudfoundry.org
[11] http://www.vmware.com/products/vsphere
[12] http://www.openstack.org

CHOReOS Enactment Engine focuses on the last phases and provides a PaaS platform for the deployment of complex service compositions, enabling the automation of the planning, preparation, and launching phases, considering that the installation and part of the configuration phases were already performed.

Fig. 2 illustrates the global view of the Enactment Engine. The deployer sends to the Enactment Engine a declarative description of the composition (e.g., a complex choreography) to be deployed, including the locations and types of the service packages. The Enactment Engine deploys these packages in one or more cloud environments and sends back, to the client, data describing the services URIs and the nodes on which each service was deployed. The choreography description can be either manually written by the deployer, or generated in a more automated way, as performed in the context of the CHOReOS project and depicted in the top of the Fig. 2; in this latter case, an automated Synthesis Process [24] reads a high-level choreography specification and generates the input to the Enactment Engine. The Enactment Engine also provides facilities to modify an already running choreography by accepting an updated choreography description. This allows the creation of automated mechanisms to deal with scaling the choreography according to the demand.

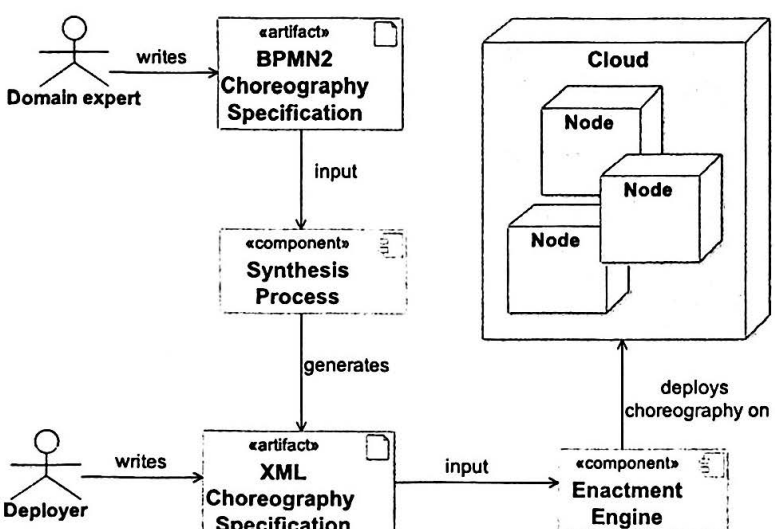

Fig. 2. Global view of the Enactment Engine

3.1 Enactment Engine components

Fig. 3 shows the Enactment Engine components. The Service Composition Deployer and the Deployment Manager are components provided by the Enactment Engine, whereas the Chef components and the Cloud Gateway are third-party software used by the Enactment Engine.

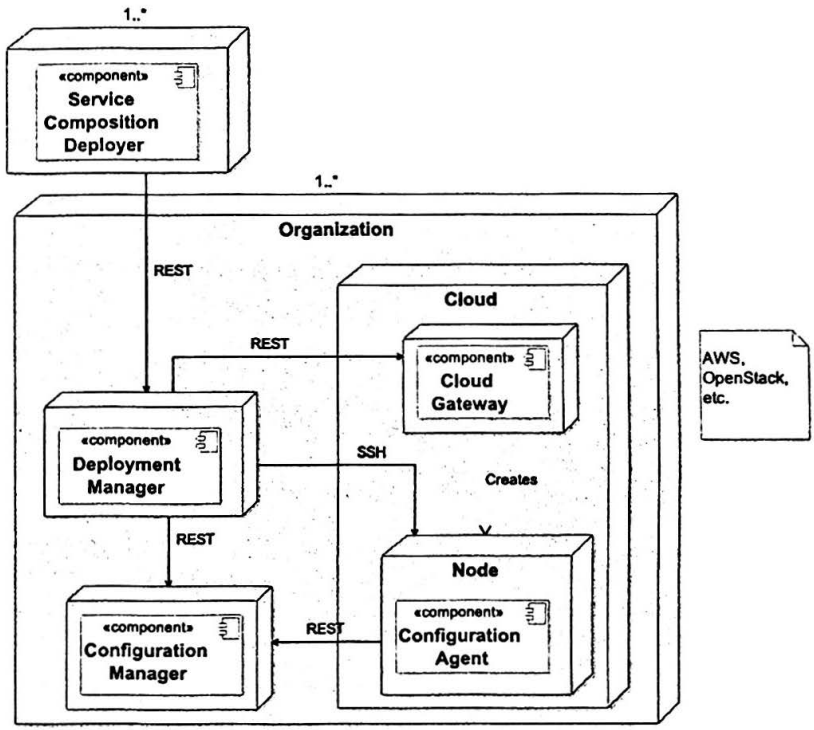

Fig. 3. Enactment Engine components

Cloud Gateway creates and destroys virtual machines (also called *nodes*) in a cloud computing environment. This component is used by the Deployment Manager, which decides when create or destroy the nodes. Currently, Amazon EC2 and OpenStack are supported as cloud gateways, but the Deployment Manager can be easily extended to support other platforms.

Chef Server stores Chef recipes and associations between recipes and nodes. "Recipes" are scripts that implement the process of configuring the operating

system, installing required middleware, and launching the services. Recipes are written in a Domain Specific Language that provides specific features for software deployment, such as idempotent actions.

Chef Client is installed by the Enactment Engine in each managed node. The Chef Client uses the Chef Server REST API to retrieve the recipes that are associated with the local node and updates the node software configuration by executing the retrieved recipes.

Deployment Manager deploys services in a cloud environment. Through its *services API*, the Deployment Manager receives a declarative service specification and selects the node onto which the service will be deployed, possibly considering service non-functional requirements. The Deployment Manager converts the received specification to a Chef recipe that implements the service preparation and launching processes. Using the *nodes API* provided by the Deployment Manager, one can request the upgrade of a node, which consists of running the Chef Client on the specified node, and thus deploying services on the node. Each Deployment Manager is associated with a Chef Server and an IaaS account.

Service Composition Deployer exposes the *choreographies API* to provide support for the automated deployment of service choreographies or orchestrations (an orchestration here can be seen as a simplified choreography in which coordination is centralized). The Service Composition Deployer client must provide the choreography declarative specification, which contains the choreography architectural description and the locations of service packages. Based on this specification, the Service Composition Deployer coordinates invocations to the multiple Deployment Managers belonging to the different participant organizations. When services are already running, the Service Composition Deployer invokes consumer services, injecting on them the addresses of their dependencies.

In addition to the services and their supporting software – such as component containers like the Apache Tomcat – the Enactment Engine also installs some management infrastructure middleware on nodes. As part of the monitoring system used in CHOReOS [25], Ganglia [26] is installed and acts as a collector of resource utilization data, such as the amount of free memory and the percentage of CPU usage. A daemon provided by the Enactment Engine is responsible for collecting measured metrics values that are considered atypical (e.g., because they exceed predefined thresholds). Moreover, to monitor running service compositions at the service level, nodes of EasyESB[13], a distributed service bus, intercept messages and gather service performance data, such as response time.

As already mentioned, the *choreographies API* receives the specification of the choreography deployment in a declarative format. Such specification is the architectural description of the choreography, which must adhere to the model defined in Fig. 4. A choreography specification provides all the needed information to enable the automated deployment of the choreography. After the deployment,

[13] https://research.petalslink.org/display/easyesb

the Enactment Engine sends back to the requesting client a response with data about the deployment status, and information about how the services can be accessed. This information is also specified according to the data model presented in Fig. 4. As shown in the figure, for each service, the specification provides important information, such as where the service package can be downloaded from (`LegacyService`), or the type of the package (`PackageType`). For example, if the package type is TOMCAT, the Enactment Engine assumes that the package is a WAR file and that it must be deployed on a Tomcat instance, which the Enactment Engine, then, deploys before deploying the service itself. The service specification also informs the initial number of replicas (`numberOfInstances` in `DeployableServiceSpec`), so the Enactment Engine will deploy each replica in a different node to increase the overall throughput. Another import service attribute is the resource impact (`resourceImpact` in `DeployableServiceSpec`), which specifies non-functional requirements of the service and that may be used by the Enactment Engine to select a suitable node to host the service.

Each service in a choreography may consume operations of other services in the choreography. Thus, each consumer service must know how to access each provider service on which it depends. Components can be bound at a variety of times: compilation, assembling, configuration and runtime [27]. When deploying services in a cloud environment, service binding must be done at runtime, since that is when the full addresses of services will be defined. As observed by Dearle [27], one option for runtime binding is using the dependency injection pattern in a similar way as proposed by Magee et al. [15, 28], in which the component receives, from the middleware, the references to its dependencies. However, Dearle [27] claims that there is a lack of frameworks for distributed dependency injection. In our solution to service binding, it is the role of the Enactment Engine to inject the dependencies the on deployed services. Thus, each consumer service must implement an operation called `setInvocationAddress` that is invoked by the Enactment Engine for each one of the service dependencies, and whose arguments provide the endpoints to the associated dependency.

Services replication associated with load balancing are common strategies nowadays to provide system scalability [12]. The Enactment Engine enables services replication within a choreography by potentially deploying multiple instances of given services and informing consumers about all of them when invoking the `setInvocationAddress` operation.

3.2 Services migration

Cloud-based large scale systems are expected to self-configure according to demand in order to cope with load peaks (by using more resources) and minimize costs (by using less resources). While not a part of the deployment process itself, this involves services migration (that is, redeploying a service onto a different node), the creation and destruction of virtual nodes, the deployment and removal of services to and from such nodes, and updating the status of consumer services (by means of the `setInvocationAddress` operation). The Enactment Engine provides an API to automate such tasks and, accordingly, the Service

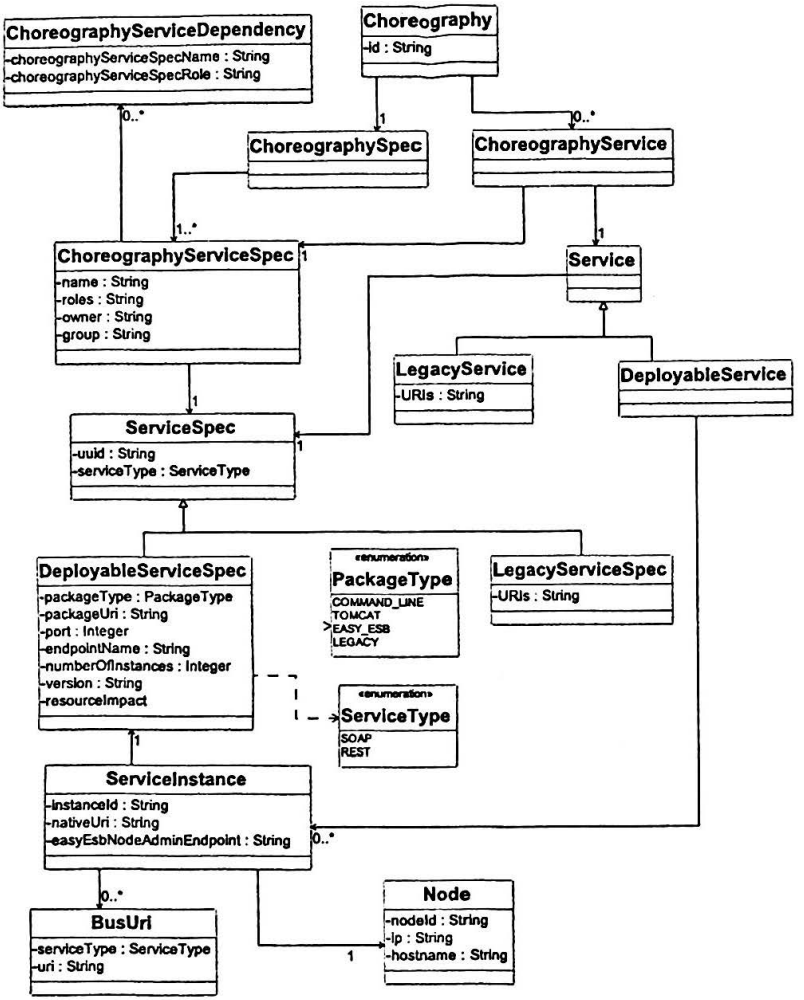

Fig. 4. Data model for choreography deployment specification

Composition Deployer offers the `updateChoreography` operation. It takes as input an updated choreography description and identifies services to be added or removed, as well as modifications in the characteristics of current services, such as the number of replicas and the expected resource impact. Based on this information, the Service Composition Deployer performs the steps necessary to

make the running choreography comply with the updated description. Typically, this involves issuing calls to the Deployment Manager providing new or updated service specifications; the Deployment Manager, in turn, may either remove a service (or some replicas of a service), deploy a new service, or redeploy a service onto nodes with different configurations (for instance, if the service needs more memory to run).

To experiment with these capabilities, we have integrated the Enactment Engine with the CHOReOS Monitoring subsystem [25]. It allows us to monitor compositions both at the service and the virtual resources levels and, in turn, to detect complex events[14], such as over- or under-loaded services and virtual machines, and take appropriate actions to handle them. These actions typically involve scaling the resources available for the composition, either by means of vertical scaling (moving a service to a more or less powerful node on the cloud) or horizontal scaling (creating more or less replicas of a service in different nodes), depending on the specific situation.

4 Implementation

In this section, we discuss some implementation aspects of the Enactment Engine that will help understand how it works, how it can provide a scalable deployment process, and how it can leverage scalable service compositions.

4.1 Extensibility

Although web services emerged to solve heterogeneity problems among systems and organizations, nowadays we have more than one mechanism to implement the concept of services. The two main approaches are SOAP services [29] and RESTful services [30]. Therefore, supporting heterogeneity is an important requirement of service-based systems.

The Enactment Engine handles heterogeneity by means of extensibility: although the current version can deploy only SOAP services packaged as JAR or WAR files, Enactment Engine users can extend it by writing a few new classes and configuration files to support new types of services. The same is true for the infrastructure cloud computing layer, since the user can write new classes to communicate with new infrastructure providers. The Enactment Engine has one more extension point to allow the creation of new policies of services per node allocation (corresponding to the planning phase of the deployment process).

4.2 REST interface

The Deployment Manager and Service Composition Deployer components provide their functionalities through a REST API. Users interact with the Enactment Engine through the Service Composition Deployer REST API by handling

[14] Complex events are events that are not identified directly, but inferred from observing their consequences (sets of other events that are identified directly).

choreographies resources, and the Service Composition Deployer uses the REST API provided by the multiple Deployment Manager instances to handle the *services* and *nodes* resources.

4.3 Service binding

As explained in Section 3, each choreography service must provide a `setInvocationAddres` operation to receive from the Enactment Engine the addresses of its dependencies. The `setInvocationAddress` arguments are the following:

dependency role: defines the operations provided by the dependency. A service may depend on multiple services with different roles, so this argument is necessary to the service know how to use the received dependency. It is a requirement that the service must to know the available operations of each role from which it depends. The role of each service must be also defined in the choreography specification, that is the Enactment Engine input.

dependency name: just a label that the dependent service may use to distinguish different available services with the same role. These different services are actually different implementations, possibly belonging to different organizations.

dependency endpoints: the list of alternative URIs to access the dependency. It has several URIs because a service may have multiple instances to improve its scalability. It is expected of the dependent service, but not required, to implement some load balancing between the different URIs. However, the dependent service may simply pick up any one of the received endpoints.

Dearle [27] alerts that forcing components to comply with the conventions of particular middleware may lock users into particular programming languages or middleware technologies. We acknowledge that forcing services to adhere the `setInvocationAddress` convention imposed by the middleware would not be desirable. However, the problems pointed out by Dearle do not apply in our case, since the Enactment Engine can be extended to support the invocation of the `setInvocationAddress` in other services types besides SOAP, and services are still reusable in other contexts not related to the Enactment Engine.

4.4 The pool of idle nodes

When a new VM is requested to the infrastructure provider, there is a chance that provisioning will fail. Moreover, some VMs may take much longer than average to be ready. In experiments conducted by us using the Amazon EC2 service, we verified that failures and long provisioning times may affect nearly 10% of the requests. Nonetheless, this is not a problem regarding only Amazon services. Several authors advocate that components on distributed large-scale systems must expect and handle faults of third-party components [13, 14, 31]. Even if the chance of a failure in each component is small, the large number of

components and interactions increases the likelihood of failures somewhere in the system [31].

The Enactment Engine expects faults on the IaaS provider and handles this issue by using a pool of idle VMs. This pool is refilled each time a new VM is requested. Also, VMs that have been previously used but are no longer necessary are not immediately destroyed, but instead added to the pool. When a request arrives, a node is retrieved from the pool, so the chances of the client being affected by IaaS problems is decreased, since some requests will retrieve already prepared machines.

The trade-off of the pool approach is the extra cost to keep VMs running in an idle state. However, this problem is treated in the Enactment Engine by a distributed management algorithm in each node: if the node is idle for $N - 1$ minutes, where N is a threshold of time that implies additional cost, the node will send to the Enactment Engine a message requesting its deletion. So, after a time of inactivity in the Enactment Engine, the pool eventually becomes empty, being filled again only when new requests arrive. This distributed approach alleviates the need to have the Enactment Engine periodically check the status of the machines to decide whether they should be removed. This feature can be deactivated in the Enactment Engine when not necessary, as in the case of using a reliable private cloud in a controlled environment.

5 Experimental evaluation

The Enactment Engine was conceived to support the provisioning, deployment and execution of complex service compositions on large-scale distributed computing platforms. We have conducted experiments to evaluate the performance and scalability of our prototype in terms of its capability to deploy a significant number of choreographies onto a real-world cloud computing platform.

We have designed synthetic workloads for our experiments. We create workloads with different number of choreographies, where each of which will have a dedicated cloud resource for each one of its service. The idea is to stress out the number of cloud resources that must be provisioned and configured by the Enactment Engine. For this reason, the pool of idle nodes is disabled on all experiments.

The choreographies are organized in such a way that the total number of services to be deployed will be grouped two by two. Each experiment will deploy and execute concurrently a fixed number of choreographies. Each time a service receives a message, a request to another service chosen by the Enactment Engine at runtime — using the `setInvocationAddress` operation described in Section 4.3 — is made.

We have chosen to evaluate the Enactment Engine scalability using the Amazon EC2 IaaS platform as the Cloud Gateway. We execute our services on up to fifty general purpose `m1.small` instances, each of which with 1.7 GiB of RAM, one core with processing power equivalent to 1.0–1.2 GHz (one Amazon EC2 Compute Unity), and a standard Amazon virtual machine image with

the Ubuntu 12.04 GNU/Linux distribution. The Enactment Engine components (Deployment Manager and Service Composition Deployer) were executed on a local machine with 8 GB of RAM, an Intel Core i7 CPU with 2.7 GHz and GNU/Linux kernel 3.6.7.

Fig. 5 present the average completion time of the deployment the deployment of all choreographies for each workload. The figure present the average of ten executions for each experiment.

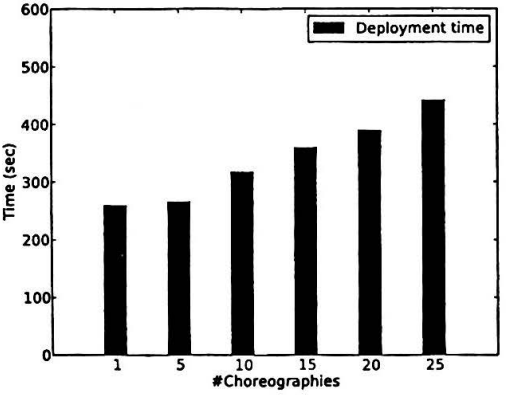

Fig. 5. Average completion time of each choreography deployment

The results suggest that Enactment Engine scales well in terms of the number of choreographies being deployed. An increase of 25 times on the number of choreographies and, therefore, an increase also of 25 times on the number of computational resources provisioned and configured by Enactment Engine, only doubled the average deployment time of the choreographies.

The workloads were designed in such a way that number of services per node was constant. However, the average completion time of the choreographies increased instead of be constant. Further investigation showed that this behavior can be partially explained by differences on the provisioning time and the occurrence of faults on the IaaS provider side.

The probability of the occurrence of delays and faults increases with the amount of computational resources concurrently requested to the IaaS provider. New experiments showed that after 1,000 requests for new nodes, 5.8% of the nodes presented an abnormal delay on the provisioning time and 0.2% had some kind of failure on the provisioning process when fifty nodes were concurrently requested over twenty iterations.

6 Conclusions

Sophisticated distributed applications of the Future Internet will be composed of a large number of highly-distributed services executing on heterogeneous mobile- and cloud-based environments, interacting at runtime time with millions of users. To enable the easy deployment and execution of such complex service compositions, flexible, robust, and adaptive middleware systems will need to be devised. In this paper, we introduced a novel middleware infrastructure supporting the adaptive enactment of complex service compositions on the cloud. This middleware highly facilitates the deployment of large-scale service choreographies on the cloud and provides runtime support for monitoring and adaptation of the service compositions.

Experimental results demonstrate that the proposed architecture is feasible and that acceptable performance can be obtained, although a few optimizations are still under way. Our source code is publicly available as open source software [15] and other researchers are welcome to contact us either to collaborate with the project or to carry out related experiments and research based on our code base.

6.1 Ongoing and Future Work

Our work in the Enactment Engine is currently geared to address the specific needs of the CHOReOS project; however, it already offers new research opportunities in several areas. The current implementation depends on the scalability of the Chef server; we intend to investigate means of leveraging the *chef-solo* tool (part of the Chef suite) to bypass the server, making the deployment process (almost) entirely distributed. We have also considered modifying both the Service Composition Deployer and the Deployment Manager to make them able to split their workloads among multiple replicas, either in terms of load-balancing requests or by creating "worker" machines to handle the actual deployment task. One interesting possibility would be to modify the Enactment Engine in order to distribute data about nodes, services, and choreographies among the cloud nodes themselves, using some sort of distributed, peer-to-peer data store (such as distributed hash tables).

We would also like to experiment with different and more sophisticated VM pool algorithms as well as to adaptively modify the size of the pool in order to strike different types of balance between deployment time and additional cost for the extra VMs according to user-defined policies and runtime status. Research on resource allocation algorithms might also benefit from using the Enactment Engine's Node Selector interface as a testbed, allowing researchers to more easily perform actual experiments with such algorithms on top of large-scale cloud environments.

In the current Enactment Engine implementation, modifying a composition that is already running works correctly only in the case of stateless services or

[15] http://forge.ow2.org/projects/choreos

services that share state. Some business processes, however, may benefit from stronger transaction semantics. We would like to investigate how Enactment Engine could be modified to empower developers and researchers to implement different strategies to handle stateful services and interactions. This mechanism should make it easy to define new policies and algorithms to handle state, such as supporting "hot" migration, forcing transaction rollbacks, or waiting for ongoing transactions to finish before deleting services.

When multiple replicas of a service are available, the Enactment Engine informs the consumer services of all available replicas and it is up to the consumer services to balance the load among them. The `setInvocationAddress` operation might be enhanced to permit different weights to be defined for each replica. A more interesting proposition would be to have the Enactment Engine itself inform each consumer about only a subset of the available replicas of a provider service to each consumer in order to split the load on them independently from the consumer services.

As mentioned, we handle dynamic adaptation by using the CHOReOS monitoring subsystem. We intend to explore the different kinds of event to detect, as well as different approaches to resource allocation while taking into account QoS parameters. Initially, we intend to do this by exploring custom rules written for the rules engine bundled within the CHOReOS monitoring system, but other approaches, such as the use of machine learning techniques and the automatic optimization and adaptation of the middleware itself may be tackled in the future.

References

1. Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F.: Service-oriented computing: State of the art and research challenges. Computer **40**(11) (2007) 38–45
2. Nanda, M.G., Chandra, S., Sarkar, V.: Decentralizing execution of composite web services. In: Proceedings of the 19th annual ACM SIGPLAN conference on object oriented programming, systems, languages, and applications (OOPSLA '04), ACM (2004) 170–187
3. Barker, A., Walton, C.D., Robertson, D.: Choreographing Web Services. IEEE Transactions on Services Computing **2**(2) (2009) 152–166
4. Chatel, P., Leger, A., Lockerbie, J.: Deliverable D6.1. Requirements and scenarios for the "Passenger-friendly airport". Available online at: http://choreos.eu/bin/Download/Deliverables (October 2011)
5. Issarny, V., Georgantas, N., Hachem, S., Zarras, A., Vassiliadist, P., Autili, M., Gerosa, M., Hamida, A.: Service-oriented middleware for the future internet: state of the art and research directions. Journal of Internet Services and Applications **2**(1) (2011) 23–45
6. Zahariadis, T., Papadimitriou, D., Tschofenig, H., Haller, S., Daras, P., Stamoulis, G., Hauswirth, M.: Towards a future internet architecture. In: The Future Internet. Volume 6656 of Lecture Notes in Computer Science. Springer (2011) 7–18
7. Steen, M., Pierre, G., Voulgaris, S.: Challenges in very large distributed systems. Journal of Internet Services and Applications **3**(1) (2012) 59–66

8. Dolstra, E., Bravenboer, M., Visser, E.: Service configuration management. In: Proceedings of the 12th international workshop on Software configuration management (SCM '05), ACM (2005) 83–98
9. Humble, J., Farley, D.: Continuous Delivery. Addison-Wesly (2011)
10. Mell, P., Grance, T.: The NIST definition of cloud computing. NIST Special Publication SP 800-145, National Institute of Standard and Technology (NIST) (September 2011) http://csrc.nist.gov/publications/PubsSPs.html#800-145.
11. Zhang, Q., Cheng, L., Boutaba, R.: Cloud computing: state-of-the-art and research challenges. Journal of Internet Services and Applications 1(1) (2010) 7–18
12. Tavis, M., Fitzsimons, P.: Web Application Hosting in the AWS Cloud: Best Practices. Technical report, Amazon (September 2012)
13. Hamilton, J.: On designing and deploying internet-scale services. In: Proceedings of the 21st Large Installation System Administration Conference (LISA '07), USENIX (2007) 231–242
14. Helland, P., Campbell, D.: Building on quicksand. CoRR abs/0909.1788 (2009)
15. Magee, J., Kramer, J.: Dynamic structure in software architectures. In: Proceedings of the 4th ACM SIGSOFT symposium on Foundations of software engineering (SIGSOFT '96), ACM (1996) 3–14
16. Balter, R., Bellissard, L., Boyer, F., Riveill, M., Vion-Dury, J.Y.: Architecturing and configuring distributed application with Olan. In: Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '98), Springer (1998) 241–256
17. Claudel, B., Huard, G., Richard, O.: TakTuk, adaptive deployment of remote executions. In: Proceedings of the 18th ACM international symposium on High performance distributed computing. HPDC '09, New York, NY, USA, ACM (2009) 91–100
18. Magee, J., Tseng, A., Kramer, J.: Composing distributed objects in CORBA. In: Proceedings of the Third International Symposium on Autonomous Decentralized Systems, 1997. ISADS 97. (1997) 257–263
19. Quéma, V., Balter, R., Bellissard, L., Féliot, D., Freyssinet, A., Lacourte, S.: Asynchronous, hierarchical, and scalable deployment of component-based applications. In: Component Deployment. Volume 3083 of Lecture Notes in Computer Science. Springer (2004) 50–64
20. Bolze, R., Cappello, F., Caron, E., Daydé, M., Desprez, F., Jeannot, E., Jégou, Y., Lanteri, S., Leduc, J., Melab, N., Mornet, G., Namyst, R., Primet, P., Quetier, B., Richard, O., Talbi, E.G., Touche, I.: Grid'5000: A large scale and highly reconfigurable experimental grid testbed. International Journal of High Performance Computing Applications 20(4) (2006) 481–494
21. Jeanvoine, E., Sarzyniec, L., Nussbaum, L.: Kadeploy3: Efficient and scalable operating system provisioning. USENIX ;login: 38(1) (February 2013) 38–44
22. Carpentier, J., Gelas, J.P., Lefèvre, L., Morel, M., Mornard, O., Laisné, J.P.: CompatibleOne: Designing an energy efficient open source cloud broker. In: Proceedings of the Second International Conference on Cloud and Green Computing, IEEE (November 2012) 199–205
23. OMG: Deployment and configuration of component-based distributed applications (DEPL) (April 2006) http://www.omg.org/spec/DEPL.
24. Autilli, M., di Ruscio, D., di Selle, A., Inverardi, P., Tivoli, M.: A model-based synthesis process for choreography realizability enforcement. In: 16th International Conference on Fundamental Approaches to Software Engineering (FASE). (2013)

25. Ben Hamida, A., Bertolino, A., Calabrò, A., Angelis, G., Lago, N., Lesbegueries, J.: Monitoring service choreographies from multiple sources. In Avgeriou, P., ed.: Software Engineering for Resilient Systems (SERENE 2012). Volume 7527 of Lecture Notes in Computer Science. Springer (2012) 134–149
26. Sacerdoti, F.D., Katz, M.J., Massie, M.L., Culler, D.E.: Wide area cluster monitoring with Ganglia. In: Proceedings of the 2003 IEEE International Conference on Cluster Computing, IEEE (2003) 289–298
27. Dearle, A.: Software deployment, past, present and future. In: Proceedings of Future of Software Engineering. FOSE '07, IEEE (May 2007) 269–284
28. Magee, J., Dulay, N., Kramer, J.: A constructive development environment for parallel and distributed programs. In: Proceedings of 2nd International Workshop on Configurable Distributed Systems, 1994. (1994) 4–14
29. W3C: Web services architecture (February 2004) http://www.w3.org/TR/ws-arch.
30. Fielding, R.T.: Architectural styles and the design of network-based software architectures. PhD thesis, University of California (2000)
31. Pollak, B., ed.: Ultra-Large-Scale Systems: The Software Challenge of the Future. Software Engineering Institute, Carnegie Mellon University (June 2006)

RELATÓRIOS TÉCNICOS

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
Instituto de Matemática e Estatística da USP

A listagem contendo os relatórios técnicos anteriores a 2009 poderá ser consultada ou solicitada à Secretaria do Departamento, pessoalmente, por carta ou e-mail (mac@ime.usp.br).

VANESSA SABINO E FABIO KON
*LICENÇAS DE SOFTWARE LIVRE HISTÓRIA E CARACTERÍSTICAS**
RT-MAC-2009-01 - Março 2009, 40 pp.

ALEXANDRE NOMA, ANA B. V. GRACIANO, ROBERTO M. CÉSAR JR., LUIS A. CONSULARO AND ISABELLE BLOCH
INEXACT GRAPH MATCHING FOR SEGMENTATION AND RECOGNITION OF OBJECT PARTS
RT-MAC-2009-02 – Março 2009, 31 pp.

CLAUDIA J. ABRO DE ARAJO AND FLÁVIO S. CORRÊA DA SILVA
GOVERNMENTAL VIRTUAL INSTITUTIONS
RT-MAC-2009-03 – junho 2009, 19 pp.

CRISTINA GOMES FERNANDES AND RAFAEL CRIVELLARI SALIBA SCHOUERY
ALGORITMOS DE APROXIMAÇÃO E PROBLEMAS COM SEQÜÊNCIAS
RT-MAC-2009-04 – agosto 2009, 38 pp.

MARCELO FINGER E GLAUBER DE BONA
UMA CONJECTURA REFUTADA SOBRE SATISFAZIBILIDADE PROBABILÍSTICA
RT-MAC-2009-05 – dezembro 2009, 18 pp.

ALEXANDRE MATOS ARRUDA E MARCELO FINGER
CARACTERIZAÇÃO DA INDEPENDÊNCIA CONDICIONAL EM LÓGICA MODAL
RT-MAC-2010-01 – Janeiro 2010, 9 pp.

FLÁVIO SOARES CORREA DA SILVA
JAM SESSION – KNOWLEDGE-BASED INTERACTION PROTOCOLS FOR INTELLIGENT INTERACTIVE ENVIRONMENTS
RT-MAC-2010-02 – Fevereiro 2010, 23 pp.

FLÁVIO SOARES CORREA DA SILVA, CLAUDIA J. A. DE ARAÚJO, LISNEY ALBERTI, ROSA ALARCON, CARLA VAIRETTI AND JESUS BELLIDO
TIMESAVER – VIRTUAL WORLDS AND ACTIVE WORKFLOWS TO DELIVER FRIENDLY PUBLIC SERVICES
RT-MAC- 2010-03 –Fevereiro 2010, 17 pp

FLÁVIO SOARES CORREA DA SILVA
NESTED INSTITUTIONS – AN ORGANIZATIONAL DESIGN PATTERN TO OPTIMIZE DISTRIBUTED WOKFLOWS IN ELETRONIC GOVERNMENT
RT-MAC- 2010-04 –Fevereiro 2010, 20 pp.

FELIPE M. BESSON, PEDRO M.B. LEAL AND FABIO KON
TOWARDS VERIFICATION AND VALIDATION OF CHOREOGRAPHIES
RT-MAC- 2011-01 – Janeiro 2011, 20 pp.

GUSTAVO ANSALDI OLIVA, FERNANDO HATTORI,LEONARDO ALEXANDRE FERREIRA LEITE AND MARCO AURÉLIO GEROSA
WEB SERVICESCHOREOGRAPHIESADAPTATION: A SYSTEMATIC REVIEW
RT-MAC-2011-02 – Janeiro 2011, 59 pp.

KELLY ROSA BRAGHETTO, JOÃO EDUARDO FERREIRA AND JEAN-MARC VINCENT
FROM BUSINESS PROCESS MODEL AND NOTATION TO STOCHASTIC AUTOMATA NETWORK
RT-MAC-2011-03 – Fevereiro 2011, 22 pp.

CLAUDIO ANTONIO PEANHO, HENRIQUE STAGNI AND FLAVIO SOARES CORREA DA SILVA
SEMANTIC INFORMATION EXTRACTION FROM SCANNED IMAGES OF COMPLEX DOCUMENTS
RT-MAC-2011-04 – junho 2011,19 pp.

FELIPE M. BESSON AND FABIO KON
"REHEARSAL: A FRAMEWORK FOR AUTOMATED TESTING OF CHOREOGRAPHIES"
RT-MAC-2011-05 – dezembro 2011, 43 pp.

PAULO H. FLORIANO, LUCIANA ARANTES AND ALFREDO GOLDMAN
CONDIÇÕES DE CONECTIVIDADE DE ALGORITMOS DE EXCLUSÃO MÚTUA EM REDES DINÂMICAS
RT-MAC-2012-01 – fevereiro 2012, 21 pp.

VIVIANE SANTOS AND ALFREDO GOLDMAN
FOSTERING INTER-TEAM KNOWLEDGE SHARING EFFECTIVENESS IN AGILE SOFTWARE DEVELOPMENT
RT-MAC-2012-02 – abril 2012, 42 pp.

CLAUDIA DE O. MELO; VIVIANE A. SANTOS; HUGO CORBUCCI; EDUARDO KATAYAMA; ALFREDO GOLDMAN; FABIO KON.
MÉTODOS ÁGEIS NO BRASIL: ESTADO DA PRÁTICA EM TIMES E ORGANIZAÇÕES
RT-MAC-2012-03 – maio 2012, 13 pp.

MIRTHA LINA FERNÁNDEZ VENERO AND FLÁVIO SOARES CORRÊA DA SILVA
*STUDYING THE BEHAVIOR OF JAMSESSION INTERACTION PROTOCOLS USING SPIN**
RT-MAC-2012-04 – junho 2012 – 27pp.

MIRTHA LINA FERNÁNDEZ VENERO AND FLÁVIO SOARES CORRÊA DA SILVA
A FORMAL SEMANTICS FOR THE JAMSESSION COORDINATION PLATFORM
RT-MAC-2012-05 – junho 2012 – 27pp.

VIVIANE ALMEIDA DOS SANTOS, ALFREDO GOLDMAN VEL LEJBMAN, DÉBORA VASCONCELOS MARTINS, HERNESTO NÓBREGA BORGES FILHO E MARIELA INÉS CORTÉS.
COMPARTILHAMENTO DE CONHECIMENTO ENTRE EQUIPES ÁGEIS: FATORES INFLUENCIADORES
RT-MAC-2012-06 – agosto 2012 – 14pp.

FLÁVIO SOARES CORRÊA DA SILVA, DAVID S. ROBERTSON AND WAMBERTO VASCONCELOS
EXPERIMENTAL INTERACTION SCIENCE
RT-MAC-2013-01 – fevereiro 2013 - 13 pp.

LEONARDO LEITE, NELSON LAGO, THIAGO FURTADO, CARLOS EDUARDO MOREIRA, DANIEL CORDEIRO, DANIEL BATISTA, MARCO AURÉLIO GEROSA, AND FABIO KON
AN ADAPTIVE ENACTMENT ENGINE FOR COMPLEX SERVICE COMPOSITIONS ON THE CLOUD
RT-MAC-2013-02 – fevereiro 2013 - 23pp.