Dynamic Reconfiguration of Cloud Application Architectures

Miguel Zúñiga-Prieto[1,2], Javier González-Huerta[3], Emilio Insfran[1], Silvia Abrahão[1]

[1]Department of Information Systems and Computation, Universitat Politècnica de València, Camino de Vera, s/n, 46022, Valencia, Spain
{mzuniga, einsfran, sabrahao}@disc.upv.es

[2]Department of Computer Science, Universidad de Cuenca, Av. 12 de Abril y Agustín Cueva, 01.01.168, Cuenca, Ecuador
miguel.zunigap@ucuenca.edu.ec

[3]Software Engineering Research Lab, Blekinge Institute of Technology, 371 79 Karlskrona, Sweden
javier.gonzalez.huerta@bth.se

Abstract:

Service-based cloud applications are software systems that continuously evolve to satisfy new user requirements and technological changes. This kind of applications also require elasticity, scalability and high-availability, which means that deployment of new functionalities or architectural adaptations to fulfill Service Level Agreements (SLAs) should be done while the application is in execution. Dynamic architectural reconfiguration is essential to minimize system disruptions while new or modified services are being integrated into existing cloud applications. Thus, cloud applications should be developed following principles that support dynamic reconfiguration of services, and also tools to automate these reconfigurations at runtime are needed. This paper presents an extension of a model-driven method for dynamic and incremental architecture reconfiguration of cloud services that allows developers to specify new services as software increments, and the tool to generate the implementation code for the services integration logic and the deployment and architectural reconfiguration scripts specific to the cloud environment in which the service will be deployed (e.g., Microsoft Azure). We also report the results of a quasi-experiment that empirically validate our method. It was conducted to evaluate their *perceived ease of use*, *perceived usefulness*, and *perceived intention to use*. The results show that the participants perceive the method to be useful and they also expressed their intention to use the method in the future. Although further experiments must be carried out to corroborate these results, the method has proven to be a promising architectural reconfiguration process for cloud applications in the context of agile and incremental development processes.

Keywords: cloud architecture, dynamic reconfiguration, service oriented architecture, model driven development, empirical validation.

1 INTRODUCTION

Service-based applications are software systems made up of software components produced either in an internal development process or provided by third parties. The architecture of a service-based application differs from the architecture of traditional software systems mainly because of its fundamental architectural element, the service; which encapsulate business functionalities and act as components from a business perspective. From a development point of view, service-based applications are mainly developed using an incremental/iterative development process [1], where the integration of new software increments not only provides the application with new functionalities but also changes (reconfigure) the current application architecture. If the integration of software increments is handled off-line, the whole system should be stopped, updated, and finally restarted. Therefore, in elastic, highly-available cloud environments, dynamic reconfiguration is required for managing architectural modifications at run-time in order to minimize system disruptions.

Cloud applications are service-based software systems, typically composed of Web services that run on cloud computing environments and consume their resources (e.g., execution environment, storage, computing, message queue services). In this context, elasticity, is one of the distinguishable characteristics of the cloud computing environments [2], which allows services acquiring more resources during a peak of demand and releasing them once they are no longer required. Furthermore, adopting a cloud computing platform introduce additional technical challenges: applications that will be deployed in cloud environments must be developed using cloud-specific APIs, project structures, and even provisioning/deployment services, thus preventing developers from creating portable services that could be redeployed in another cloud environment with regard to a given Quality of Service (QoS) level or a Service Level Agreement (SLA). Not only approaches to support developers to design, implement, and deploy software systems are required [3]; but also architectural approaches to cope with interoperability and portability of services [4]. Furthermore, in terms of architectural reconfiguration, as far as we know, there are no proposals that support a systematic reasoning about the architectural impact of the integration of services included in a given software increment into the current application architecture.

Even though the dynamic reconfiguration of software architectures occur at runtime, it should be supported along the different stages of the development process from the very beginning. A Model-Driven Development (MDD) approach may provide good support to automate the dynamic reconfiguration of cloud application architectures. MDD may also overcome portability issues by modeling the cloud application architectures at different levels of abstractions and then to obtain implementation/deployment artifacts by means of model to model and model to code transformations.

In previous works [5],[6], we introduced the main ideas of the process definition for the DIARy method to support the specification and generation of some software artifacts for service architecture reconfigurations. The DIARy method follows an incremental and MDD approach that supports the incremental integration of cloud service applications and their dynamic architecture reconfiguration triggered by the integration of new *software increments* (hereafter referred to as increments). In this paper, we extend the DIARy method by redefining and defining new activities and tasks to support the building, packing, provisioning, and deployment of cloud services. For example, new activities were defined in order to allow developers not only to organize services into projects that can be built, packed and deployed as an independent deployment artifact in order to share cloud environment resources, but also to specify the cloud environment resources (infrastructure and platform) needed for the deployment. We also provide the tool support to automate these tasks by defining the metamodels, which define the service architecture and the cloud resources needed to deploy services, as well as the transformation chains, which automate the generation of software artifacts that implement the integration logic (orchestration among services), scripts for infrastructure/platform provisioning and deployment, and scripts for architectural reconfiguration that change service invocations according to the integration specification. Finally, this paper also reports the results of a quasi-experiment carried out by 20 participants, including PhD and Master's computer science students, in which we analyze the *perceived ease of use*, *perceived usefulness*, and *intention to use* of the participants in using the DIARy method and its corresponding tool.

The remainder of this paper is structured as follows: Section 2 discusses the related works. Section 3 presents an overview of the method proposed. Section 4 illustrates the use of our method on a running example. Section 5 presents the preliminary results of the validation of the method through a quasi-experiment. Finally, Section 6 presents our conclusions and final remarks.

2 RELATED WORK

In this section we discuss approaches that support the development of cloud applications as well as the dynamic architecture reconfiguration. There exist some development approaches that apply MDD principles in order to tackle portability issues when developing or migrating cloud applications (e.g., [7], [8], [9]). With regard to approaches that propose mechanisms with which to document design decisions in cloud environments we can highlight CAML [10], MULTICLAPP [11] and CloudML [12]. These works define UML profiles or other modeling languages used to describe deployment topologies, applications as a composition of software artifacts to be deployed across multiple clouds, or resources that a given application may require from existing clouds. Additionally proposal such as Chef [13] or [14] abstract the deployment of services from specific cloud providers and offer deployment platforms; however, these proposals create dependencies with their technology. Although "getting integration right is the single most important aspect of the technology associated with agile approaches" [15], these proposals do not provide mechanisms with which to specify architectural decisions regarding integration and the impact of integrating increments in the current cloud application architecture.

With regard to proposals that support the dynamic architecture reconfiguration, Breivold et al. [4] conducted recently a systematic review on architecting for the cloud. They categorized studies that describe architectural approaches and design considerations when architecting for the cloud. The authors identified the need for design/architectural approaches for supporting the maintenance of cloud services. The EU project SeaClouds [16], proposes a platform to performs seamless adaptive multi-cloud management of service-based applications and dynamically reconfigures them by changing the orchestration (interaction coordination) of services depending on monitoring results. The MODAClouds [9] project is another research that focuses on the implementation of a framework with which to develop and deploy applications in multi-clouds, in which monitoring triggers adaptation actions such as the migration of system components from one cloud to another, along with the dynamic re-deployment of the final application or its components. Despite the fact that the reconfiguration takes place by replacing orchestration or as result of the re-deployment of components, these proposals take into account alternatives as regards provisioning and deployment, they do not focus on software architecture aspects nor take into account implementation alternatives that facilitate scalability and architectural reconfiguration. What is more, in the aforementioned proposals, the reconfiguration/re-deployment starts as the result of monitoring activities; however, adaptive changes (e.g., integration of software increments resulting from new functionalities) that require architectural reconfiguration are not taken into account.

According to [17] architecture centric reconfiguration proposals included some formal proofs as part of the evaluation, evidence is often obtained from applying the approach to small case studies and examples, and only a few empirical studies have been undertaken; however none of them is related to the cloud environment. With regard to empirical validation of the before identified related proposals (SeaClouds, and MODAClouds), as far as we know, they do not present empirical validations; therefore a lack of frameworks and standard criteria for the comparison of dynamic architecture reconfiguration processes triggered by new services integration. For instance, MODAClouds uses an example [9] in order to provide intuition on the benefits of the proposal. However in software architecture field more empirical studies to build the body of knowledge.

3 THE DIARY METHOD

In this section, we briefly describe the DIARy method (see Figure 1), a model-driven approach for the dynamic reconfiguration of cloud application architectures caused by the integration of software increments. This method allows developers to specify how the services included in an increment will be integrated into a cloud application which is already deployed. In this work, we redefined two out of the three the activities of the DIARy method. The *Increment Implementation* activity was redefined to explicitly deal with the increment integration specification to generate software artifacts corresponding to the implementation of the integration logic (e.g., services orchestration, interaction protocol), and scripts with which to build, deploy and architecturally reconfigure the current cloud application, all of which are generated according to the cloud environment where services will be deployed.

We also redefined the activity *Deployment & Architectural Reconfiguration* to deal with the specification of cloud resources that services need to be deployed, and the generation of the corresponding provisioning and deployment scripts. As a consequence of these changes the DIARy new activities are shown in Figure 1.

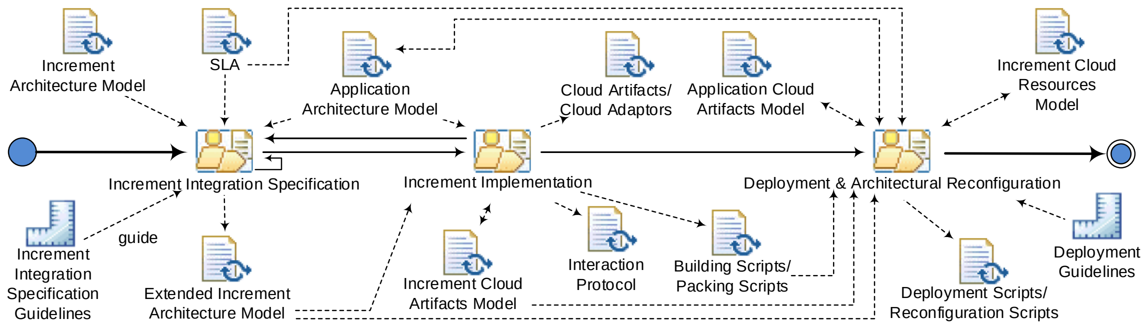
Increment Architecture Model
SLA
Application Architecture Model
Cloud Artifacts/ Cloud Adaptors
Application Cloud Artifacts Model
Increment Cloud Resources Model
Increment Integration Specification
Increment Implementation
Deployment & Architectural Reconfiguration
guide
Increment Integration Specification Guidelines
Extended Increment Architecture Model
Increment Cloud Artifacts Model
Interaction Protocol
Building Scripts/ Packing Scripts
Deployment Scripts/ Reconfiguration Scripts
Deployment Guidelines

Figure 1. The *DIARy process*

3.1 Increment Integration Specification

The activity is aimed at supporting the systematic reasoning about the integration logic and the architectural impact of integrating the services included in a software increment into the current cloud application (*Application Architecture Model*), regardless the cloud environment. In this activity, developers: i) take as input the architectural design of a software increment (*Increment Architecture Model - IAM*) described with the Service oriented architecture Modeling Language (SoaML) [18], an OMG standard specifically designed for the modeling of service-oriented architectures. ii) Apply the *DIARy-Specification-profile* [19] to the *IAM*, extending the expressiveness of the SoaML, and providing it with features that allow software architects to specify integration logic and architectural impact. and iii) Follow *Increment Integration Specification Guidelines* to make integration design decisions based on *SLA* term; producing as output the *Extended Increment Architecture Model (EIAM)* (see ***Figure 4a*****)**. The *EIAM* complies with the *Extended Increment Architecture Model* metamodel which we defined by extending UML and SoaML metamodels using the same concepts we used to define the *DIARy-Specification-profiled,* see [19],[5] for details.

Architects use the current *Application Architecture Model* to identify the architectural elements of the current architecture which, after integration, will change or will interoperate with architectural elements of the *IAM*; then specify the integration logic and the architectural impact. Integration logic is specified by describing interoperation (*Service Contract*) among *Participants* involved in a service. *Service Contracts* inner parts include interfaces that *Participant* elements must implement in order to interoperate, as well as the interaction protocol which is described by an activity diagram. A *Participant* represents: i) a service to be integrated, ii) a service/component already existing in the current Application Architecture Model with which a service/component of the *IAM* will interoperate, and iii) a service/component to be created in order to consume or provide services.

Similarly, developers specify architectural impact by tagging every *IAM* architectural element (e.g., *ServiceContracts*, *Participants*, and dependencies among them - *RoleBindings*) with values that describe how its integration will change the current *Application Architecture Model* (e.g., Adding or removing architectural elements). (see a*rchitecturalImpact* attribute in architectural elements of ***Figure 4a***). Additionally, developers specify the requirements about the management of performance of services by providing information about the expected level of elasticity and delay in processing requests (see *elasticityLevel* and *delayLevel* attributes ***Figure 4a***). Developers use this information, in later development phases, to select either implementation/provisioning/deployment alternatives, or cloud environment resources that satisfy SLA terms or other requirements.

3.2 Increment Implementation

This activity was redefined for this work. It aims to support the integration process by generating platform-specific cloud artifacts (software artifacts to be deployed on a specific cloud environment) that implement the increment integration specification (e.g. integration logic). This activity includes the following steps:

3.2.1 Check Increment Compatibility

This step is aimed to reduce the risk of incompatibilities that avoid the integration between the software increment and the current cloud application architecture. Developers participate in verifying whether the *EIAM* is compatible with the current *Application Architecture Model*. If discrepancies exist between interfaces (e.g., different names for methods and services, different message ordering), they design a *ServiceContract* that overrides the current one and apply model-to-text (M2T) transformations that generate *Cloud Adaptors* (see **Figure 1**) that implement the new integration logic.

3.2.2 Specify the Packing and Deployment Structure

In this step, the *Increment Cloud Artifacts Model* is generated in order to describe the cloud artifacts (*Artifacts*) and cloud environment resources needed to implement *EIAM* architectural elements (i.e., *Service Contracts*, *Participants* and *Rolebindings*).

The way in which services are deployed has an influence on satisfying *SLA* terms or other nonfunctional requirements (e.g., agility to deploy, cost of provisioning) [20]; therefore, this model organizes services' related *Artifacts* into *Projects* or group of *Projects* that can be built, packed and deployed independently on different cloud environments in accordance with decisions made during the development process (e.g. implementation technology, management of performance of services, managing of running cost by deploying a group of *Projects* on a shared host). Additionally, in order to promote the decoupling of the integration/interaction logic from the *Participants* logic (business logic), and ease the architectural reconfiguration by replacing either the interaction protocol or dependencies among Participants, we propose to place *Artifacts* related to interaction into *InteractionProjects* and Artifacts related to *Participants* logic into *ImplementationProjects.*

Best practices in continuous delivery suggest storing configuration information that change at runtime outside the deployable package [21] thus enabling them to be updated without requiring the redeployment of the entire package. Therefore, in order to facilitate dynamic architectural reconfiguration by updating dependencies among services; the *Increment Cloud Artifacts Model* allows to specify services' dependencies (*EndPoints*) in configuration *Artifacts* (*DynamicConfiguration*) that will be independently deployed.

The *Increment Cloud Artifacts Model* complies with the *Cloud Artifacts Model* metamodel that we already proposed in [5]. In this work, in order to support the *Packing and Deployment Structure* specification, we provide an Eclipse plug-in which executes M2M transformations carried out using the Atlas Transformation Language (ATL) to generate *Increment Cloud Artifacts Models* from *EIAM*s. **Figure 2** (lines 3, 6 and 10) shows an example of the transformation rule applied to assign the *Artifacts* corresponding to services (e.g., *FrontEndServices*) offered by *Participants* that require a high *elasticityLevel* into an exclusive *Project*, which will be deployed on an exclusive host (e.g., a virtual machine).

3.2.3 Generate Implementation Code

In this step, architects make the implementation decisions that best fit the individual requirements of each service included in an increment and complete the previously generated *Increment Cloud Artifacts Model* by specifying: i) the *technology* in which services related *Artifacts* will be implemented (e.g., source code language); ii) service's configuration information that could change at runtime, by creating or updating metaclasses of *DynamicConfiguration* type; iii) inter-service communication information (e.g. SOAP/REST service style, message format, protocols); and iv) the *location* of *Artifacts* to be generated. Once developers have completed the *Increment Cloud Artifacts Model*, they execute M2T transformations that use this model and the *EIAM* as input in order to generate *Artifacts'* implementations according to the specified technology, which are: i) the *Interaction Protocol* between the services to be integrated and the current application, which will be offered as another service; ii) interface implementations or skeletons of service`s logic, depending on how detailed is the design of architectural elements inner parts; iii) as many configuration files as *DynamicConfiguration Environments* (e.g., development, testing, production), iv) *Building/Packing s*cripts, according to the *DeploymentProject*s*'* structure (see outputs of this activity in **Figure 1**). Finally, cloud developers complete the generated skeletons, then run the previously generated *Building/Packing s*cripts obtaining a deployment artifact.

```
rule ParticipantUse2Implementation {                -- Create a Implementation project (and related elements) per ParticipantUse
from
    ParticipantInput : eiam!ParticipantUse(
        ParticipantInput.elasticityLevel = 4 )            -- Filter elements whose elasticityLevel = high (4)
to
implementation : cam!ImplementationProject(               -- Crate an Implementation Project
    name <- ParticipantInput.name,                        -- Assign the Participant name to the Project name
    belongsToParticipant <- ParticipantInput.participantType,
    artifacts <- front,                                   -- Create an Artifact FrontEndService
    deployment <- thisModule.resolveTemp(                 -- Assign Implementation Project to Participant's Deployment project
        ParticipantInput.participantType, 'DeploymentProject')),
front : cam!FrontEndService(                              -- Create a FrontEndService element
        serviceProject <- implementation),
configimplement : cam!DynamicConfiguration(               -- Create a DynamicConfiguration element
```

Figure 2. Excerpt of M2M for generating the *Increment Cloud Artifact Model*

3.3 Deployment & Architectural Reconfiguration

This activity was redefined for this work. Once developers have released increments as deployment artifacts, they prepare artifacts to be deployed on the corresponding cloud environment(s) (e.g., production, testing). In this activity cloud developers specify cloud resource requirements and generate scripts that automate infrastructure provisioning, deployment and architectural reconfiguration.

3.3.1 Deployment

Architects make provisioning and deployment decisions, about the infrastructure and platform resources that must be provisioned in order to deploy and allow execution of the services included in a deployment artifact, and document them in the *Increment Cloud Resources Model*. The *Increment Cloud Resources Model* complies with the *Increment Cloud Resources Model* metamodel (see **Figure 3**) and allow architects to specify the *CloudEnviroments* where a *DeploymentArtifact* will be deployed. A *CloudEnvironment* is made of *Infrastructure* and *Platform* resources and architects define as many *CloudEnvironments* as cloud providers, where resource characteristics vary according to the offerings of a specific cloud provider. The *Increment Cloud Resources Model* also allows to describe *Subscription* information (e.g. *Credentials*, *Parameters*), which is used to manage the provisioning and deployment.

After specifying cloud resources requirements, cloud developers execute the M2T transformations that use as input the *Increment Cloud Resources Model* to generate *Deployment Scripts* (see outputs of this activity in **Figure 1**) not only specific for the cloud environment(s) chosen for deployment, but also according to the *architecturalImpact* specified for *Participants* and Service *Contracts* during the *Increment Integration Specification* activity. For example, scripts will include instructions to deploy packages when *architecturalImpact = Add*, and instructions to undeploy when *architecturalImpact = Delete*. Cloud providers allow developers to provision resources or deploy services by executing scripts (or using APIs). However, according to what we have experienced when deploying services, developers are not asked to provide detailed information about resources they need to provision; instead they create predefined cloud environment instances. For example, instead of creating a virtual machine (*IaaS_Computing*) with 2GB of memory, two cores and *PaaS_OS* Windows; developers create a predefined *small* virtual machine. Even though the *Increment Cloud Resources Model* describe detailed resources' characteristics, the mapping to a specific predefined instance is done when defining the transformation rules. The later avoid architects to have to know the predefined cloud environment instances offered by a cloud provider. Instead the model transformation process finds the instance that best fit the resource requirements that were specified in the Increment *Cloud Resources Model*, then generates the script to create that instance.

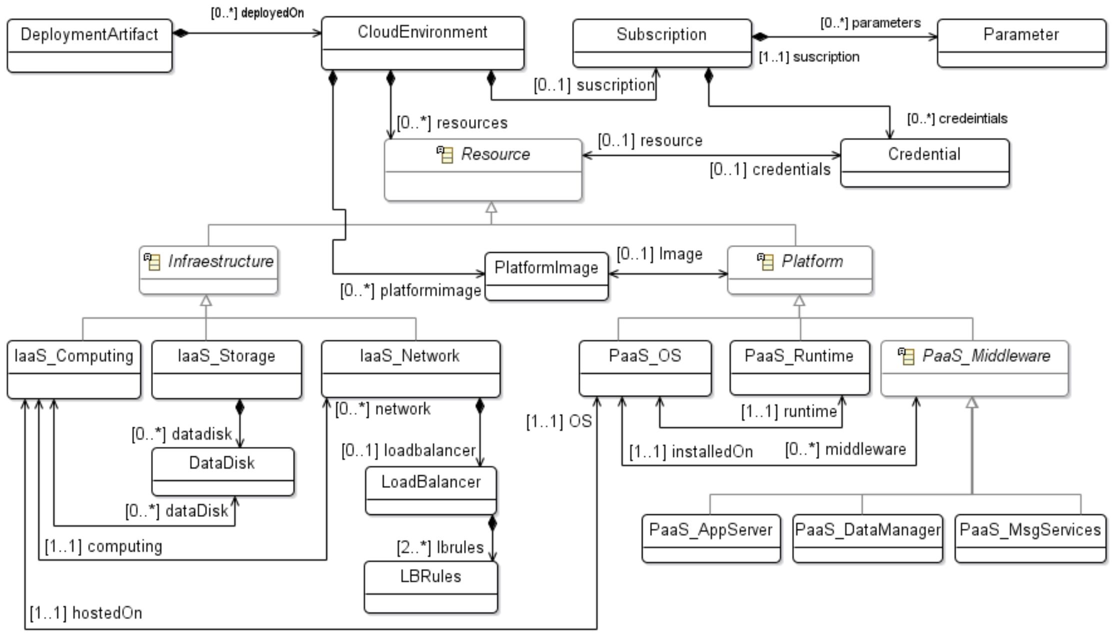

Figure 3. Excerpt of *Increment Cloud Resources Model* metamodel

3.3.2 Architectural Reconfiguration

Architectural reconfiguration is achieved by deploying/removing services, and by changing service dependencies at run time. After executing the deployment scripts the *Cloud Artifacts Model* is updated with information about *EndPoints* that will use services to expose their functionalities as well as to invoke other services. Then, cloud developers execute M2T transformations that generate scripts with which to reconfigure the application architecture, which use the *architecturalImpact* specified for *RoleBindings* during the *Increment Integration Specification* activity to dynamically update *EndPoints* information stored in the service configuration files.

Finally, the *EIAM* and the *Increment Cloud Artifacts Model* are used as the input for the M2M transformations that update both the current *Application Architecture Model* and the *Application Cloud Artifacts Model* by integrating the increment corresponding architectural elements and cloud artifact descriptions (see outputs of this activity in **Figure 1**).

4 ILLUSTRATIVE EXAMPLE

In this section, we describe the main *DIARy* artifacts and activities through an illustrative example. A manufacturing company wishes to improve the technological support given to its dealers, and is considering updating its already existing *manufacturer* service by including new functionalities which will allow dealers to place production orders and get their products shipped through a third-party shipping service. This illustrative example has been adapted from the examples included in the SoaML documentation [18], and which are widely used/adapted in proposals that refer to SoaML.

First, In the *Increment Integration Specification* activity, architects take as input the *IAM* and the current *Application Architecture Model* and generate *EIAM* as output. These models are built by using SoaML and the *DIARy-specification-profile*, respectively. The integration logic is specified by refining existing *ServiceContract* elements of the *IAM* or by creating new ones, while architectural impact is defined by tagging the architectural elements of the *IAM* according to their *architecturalImpact* on the current *Application Architecture Model* (see the training material provided in [22] for a detailed explanation). In this example, the *IAM* includes only the *Shipper* service design, therefore a *ServiceContract* that describe interaction/integration must be created.

Figure 4a shows the generated EIAM, where, in order to integrate a new service (provided by a *Shipper*) the interoperation protocol with services already existing in the current *Application Architecture Model* must be replaced with the new versions of the services. Therefore the PlaceOrder *ServiceContract* must be *deleted* and a new *ServiceContract* (PlaceOrderWithShipping) that define the integration of the new service with the already existing services must be added. Additionally the Manufacturer logic must be modified in order to implement interfaces that allow interaction with the new *Shipper* service. Finally, The *Dealer,* which is the participant who initiates the interaction must change its dependencies to point to the new interoperation/integration service (OrderWithShiping), therefore the *architecturalImpact* of its *RoleBindigs* with the actual interoperation/integration service (PlaceOrder) must be *deleted*; whereas RoleBindings with OrderWithShipping are added. The requirements of workload change management for *OrderWithShipping* are *ElasticityLevel* = High.

During *the Increment Implementation* activity developers use the Eclipse plug-in provided as support for this activity in order to execute M2M transformations that generate the *Increment Artifacts Model* (Figure 4b) from the *EIAM*, then complete this model by providing *Project* information related to the packing structure. Elasticity level requirements of Shipper and interaction/integration services[1] were *High*, therefore *Artifacts* related to each service were placed in an exclusive project and will be deployed independently. Our deployment platform was Windows Azure, therefore we use Visual Studio 2013 as the development platform. We define M2T ATL Transformations that generate *Implementation Code* corresponding to *Interaction Protocol* (Figure 4d). This transformation take as input the interaction protocol specified as a sequence diagram (described as the inner part of the *ServiceContract*) and generate as output the choreography to be deployed as an Azure workflow WCF service. With regards to configuration files, information related to EndPoints was manually modified in Visual Studio 2013 according to the *Increment Cloud Artifact Model* description.

[1] The interaction/integration among services is handled by a new *Interaction/Integration* service which is specified as *ServiceContracts* among participants.

For the deployment, we model the resources to support the increment as instances of the *Increment Cloud Resources Model* metamodel (see Figure 4c). We specify the cloud resource requirements of the Shipper and OrderWithShipping services. We generate the deployment scripts through an Acceleo (http://www.eclipse.org/acceleo/) M2T transformation that takes as input this *Increment Cloud Resources Model*.

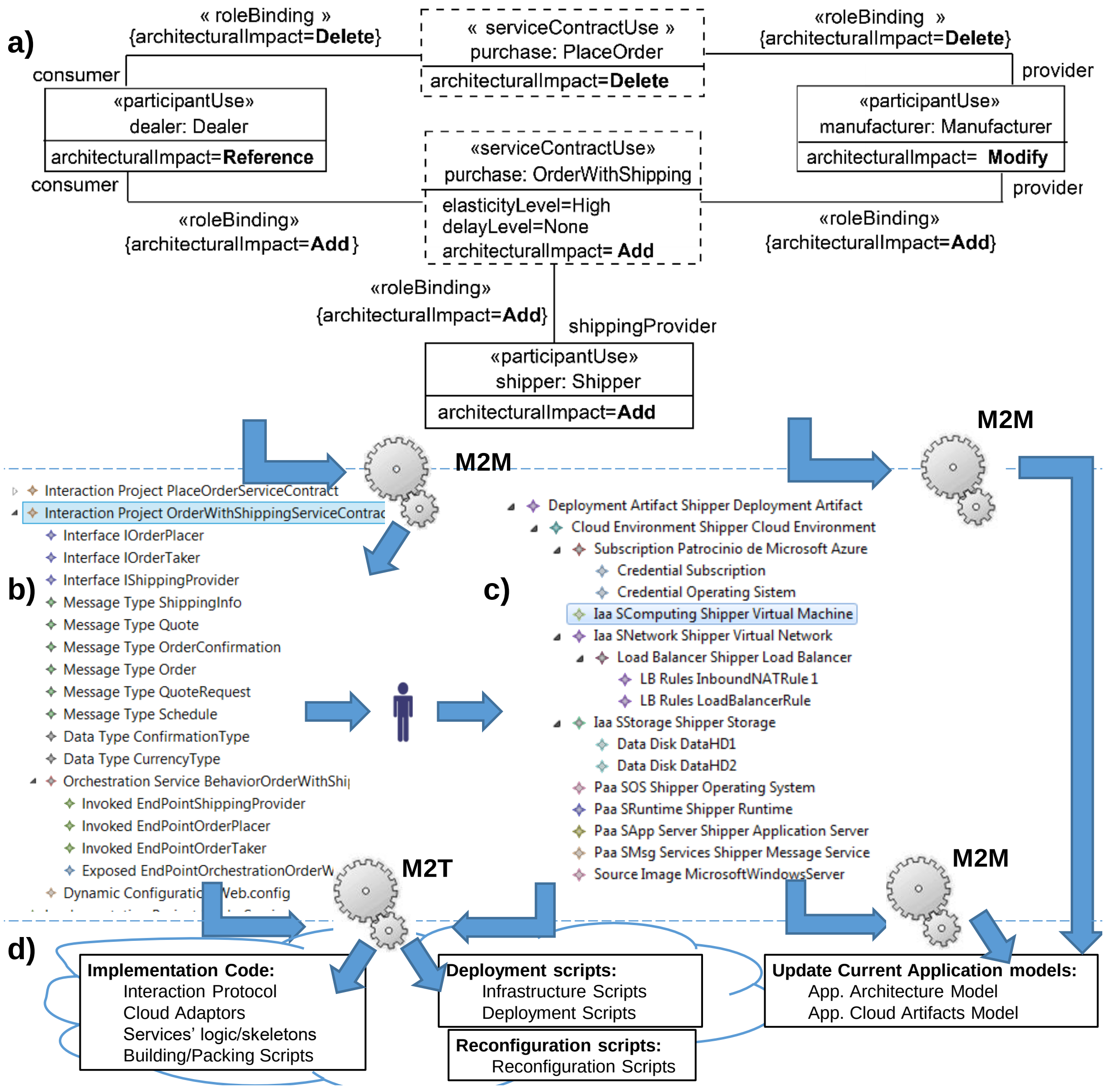
a)
« roleBinding »
{architecturalImpact=Delete}
« serviceContractUse »
purchase: PlaceOrder
architecturalImpact=Delete
«roleBinding »
{architecturalImpact=Delete}
consumer
provider
«participantUse»
dealer: Dealer
architecturalImpact=Reference
«participantUse»
manufacturer: Manufacturer
architecturalImpact= Modify
«serviceContractUse»
purchase: OrderWithShipping
elasticityLevel=High
delayLevel=None
architecturalImpact= Add
consumer
provider
«roleBinding»
{architecturalImpact=Add}
«roleBinding»
{architecturalImpact=Add}
«roleBinding»
{architecturalImpact=Add}
shippingProvider
«participantUse»
shipper: Shipper
architecturalImpact=Add
M2M
M2M
Interaction Project PlaceOrderServiceContract
Interaction Project OrderWithShippingServiceContrac
Interface IOrderPlacer
Interface IOrderTaker
Interface IShippingProvider
Message Type ShippingInfo
Message Type Quote
Message Type OrderConfirmation
Message Type Order
Message Type QuoteRequest
Message Type Schedule
Data Type ConfirmationType
Data Type CurrencyType
Orchestration Service BehaviorOrderWithShi
Invoked EndPointShippingProvider
Invoked EndPointOrderPlacer
Invoked EndPointOrderTaker
Exposed EndPointOrchestrationOrderW
Dynamic Configuratic Web.config
b)
c)
Deployment Artifact Shipper Deployment Artifact
Cloud Environment Shipper Cloud Environment
Subscription Patrocinio de Microsoft Azure
Credential Subscription
Credential Operating Sistem
Iaa SComputing Shipper Virtual Machine
Iaa SNetwork Shipper Virtual Network
Load Balancer Shipper Load Balancer
LB Rules InboundNATRule1
LB Rules LoadBalancerRule
Iaa SStorage Shipper Storage
Data Disk DataHD1
Data Disk DataHD2
Paa SOS Shipper Operating System
Paa SRuntime Shipper Runtime
Paa SApp Server Shipper Application Server
Paa SMsg Services Shipper Message Service
Source Image MicrosoftWindowsServer
M2T
M2M
d)
Implementation Code:
Interaction Protocol
Cloud Adaptors
Services' logic/skeletons
Building/Packing Scripts
Deployment scripts:
Infrastructure Scripts
Deployment Scripts
Reconfiguration scripts:
Reconfiguration Scripts
Update Current Application models:
App. Architecture Model
App. Cloud Artifacts Model

Figure 4. DIARy method main transformation chain; excerpts of: a) *Extended Increment Architecture Model*, b) *Increment Cloud Artifacts Model*, c) *Increment Cloud Resources Model*, d) generated artifacts and models update

In this example reconfiguration takes place by deploying new services and updating dependencies among services which are included in services' configuration files. During the *Architectural Reconfiguration*, we use Acceleo in order to obtain these *Reconfiguration Scripts* that update services' configuration files (see Figure 5). We generated XML Document Transform (XDT) files used in Visual Studio to modify service configuration files while the deployment takes place. Figure 5 (lines 12, 13, 14) shows an example of the transformation rule applied to generate scripts that modify configuration information related to *RoleBindings* among services in accordance with architectural impact specification (see *artifactImpact* of *RoleBindings* in Figure 4a). Finally, deployment and reconfiguration scripts were executed.

```
[template public generateElement(aCloudArtifactsModel : CloudArtifactsModel)]
[for(InteractionProjects:InteractionProject | projects->select(oclIsTypeOf(InteractionProject)))]
    [file (InteractionProjects.name.concat('/ServiceDefinicion.csdef'), false)]

<?xml version="1.0" encoding="utf-8"?>
    <ServiceConfiguration serviceName="[InteractionProjects.name/]" xmlns="…" xmlns:xdt="http://.../XML-Document-Transform" >

    [for(IK:Invoked | InteractionProjects.interactionService.endpoints->select(oclIsTypeOf(Invoked)))]
        <WebRole name="[InteractionProjects.name/]">
            <ConfigurationSettings xdt:Transform="InsertIfMissing">

            [comment parseXDT executes a mapping between artifactImpact and XDT values /]
                <Setting name="[IK.name.concat('_EndPoint')/]" [parseXDT(IK.artifactImpact)/]/>
                <Setting name="[IK.name.concat('_Binding')/]" [parseXDT(IK. artifactImpact)/]/>
```

Figure 5. Excerpt of M2T used to generate *Reconfiguration Scripts*

5 EMPIRICAL VALIDATION

This section presents a quasi-experiment, an empirical study aimed at testing descriptive causal hypotheses about manipulable causes in which units (subjects) are not assigned to conditions randomly [23], with the aim of analyzing the DIARy method for the dynamic reconfiguration of cloud architectures triggered by the integration of new services into cloud applications. One of the reasons for selecting a quasi-experiment for the validation of *DIARy* is the absence of widely accepted and integrated methods that support the dynamic reconfiguration of cloud service architectures due to adaptive changes. This fact increases the difficulty to perform controlled experiments that compare *DIARy* with other similar methods. The validation strategy is also intended to contribute to Software Engineering through a well-defined validation framework that can be reused by other researchers in the empirical validation of other architecture reconfiguration methods.

5.1 Experiment Planning

The quasi-experiment was designed following the guidelines proposed by [24]. According to the Goal-Question Metric (GQM) paradigm [25], the goal of this quasi-experiment is to **analyze** the DIARy method **with the aim** of evaluating the perceived ease of use, perceived usefulness, and intention to use of this method **from the point of view** of a group of novel software architects.

The research questions addressed by the experimentation are:

- RQ1: Is the DIARy method perceived as both easy to use and useful in the architectural reconfiguration of cloud applications?
- RQ2: Is there an intention to use the DIARy method in the future?

The context of the experiment was determined by: i) the cloud application in which cloud services will be integrated, and whose architecture will be dynamically reconfigured; ii) the DIARy method activities to be evaluated in the quasi-experiment; and iii) the subject selection. In this experiment we focus on a cloud application of a *Reservation System,* based on the example presented in [26], which allows travel agencies to manage their customers' reservations. The experimental tasks consisted in the integration of the architecture of an increment (*i.e., Increment-1)* in the existing architecture of the *Reservation System*.

The goal was to modify the cloud application architecture due to the deployment of the *Increment-1* by incorporating a new actor, modifying the business logic of the existing actors, and replacing the interaction protocol of the services in which the new actor is involved. We provided an initial version of the system which was deployed in the Microsoft Azure© cloud environment. The architecture of the system was initially composed of 5 services, 4 participants and 10 role bindings. The *IAM* (after applying the *DIARy-specification-profile* on it) corresponding to *Increment-1* consisted of 1 service, 3 participants, and 3 role bindings. We included as experimental tasks only the DIARy method tasks required to obtain the reconfiguration scripts that change links among services at runtime since this quasi-experiment context is the architectural reconfiguration of cloud applications. The participants worked with the integration scenario, performing tasks that supported both the *Increment Integration Specification* of the *Increment-1,* and the *Generation of the Reconfiguration Scripts* needed to reconfigure the *Reservation System* architecture at run-time. Once reconfigured, the behavior of the system changed, due to the addition of the new actors and the changes performed in the interaction protocol.

We focus on a novel architect profile since one of our goals is to provide a reconfiguration method suitable to support inexperienced architects while performing dynamic reconfiguration activities. The quasi-experiment was executed in an academic environment with subjects selected by convenience from two research groups at the Universitat Politècnica de València, with experience on modeling and/or web services development (their profiles are shown in Table 1) and whose participation was voluntary.

Table 1. Participant Profiles

Profile	***Description***
PRO1	PhD students all currently working on research topics related to various areas of SE.
PRO2	PhD students all currently working on research topics related to various areas of Computer Science
PRO3	Master students all currently working on research topics related to various areas of SE.
PRO4	Undergraduate students all currently working on research topics related to various areas of SE.

We defined three subjective dependent variables based on the Technology Acceptance Method (TAM) [27], which is a theoretical model for analyzing user acceptance and usage behavior of emerging information technologies [28]; its primary constructs are the following subjective variables:

- **Perceived Ease of Use (PEOU),** which refers to the degree to which evaluators believe that learning and using a particular method will be effort-free.
- **Perceived Usefulness (PU),** which refers to the degree to which evaluators believe that using a specific method will increase their job performance within an organizational context.
- **Intention to Use (ITU),** which refers to the extent to which an evaluator intends to use a particular method. This last variable represents a perceptual judgment of the method's efficacy – that is, whether it is cost-effective and is commonly used to predict the likelihood of acceptance of a method in practice.

We relied on an existing measurement instrument to measure the subjective variables, the Method Evaluation Model (MEM) [29] which is based on TAM; although we adapted it (basically by rewording statements) for use in the context of a dynamic architecture reconfiguration processes and to operationalize TAM. The measurement instrument was a 5-

point Likert scale questionnaire with a set of 15 closed questions: 5 for PEOU, 7 for PU, and 3 for ITU, formulated by using a 5-point Likert scale. The aggregated value of each subjective variable was calculated as the arithmetical mean of the answers to the questions associated with each subjective dependent variable. The order of the questions in this questionnaire was shuffled in order to prevent systemic response bias, and were formulated to become negative statements on the left-hand side to avoid monotonous responses [30].

We also defined two performance-based variables to measure subjects' actual effectiveness (number of correctly specified architectural changes (as being coherent with the integration scenario defined in the booklet) and actual efficiency (ratio between the number of correctly specified architectural changes and the total time spent on their specification.) when applying the DIARy method. Finally, the questionnaire also contained three open-questions in order to obtain feedback from the participants.

The likelihood of acceptance of a dynamic architecture reconfiguration method that support the integration of new services in practice can be predicted by testing the following hypotheses:

- $H1_0$: The DIARy method is perceived as difficult to use. $H1_1 = \neg H1_0$.
- $H2_0$: The DIARy method is perceived as not useful. $H2_1 = \neg H2_0$.
- $H3_0$: There is no intention to use the DIARy method in the future. $H3_1 = \neg H3_0$.

Other factors may have an effect on the variables under study such as the actors background: the previous experience of participants on topics related to software modeling and web services development may affect the perceptions and performance when applying the process. We defined three factors: i) modeling experience (*ExpModeling*), ii) experience on services development (*ExpServ*), and iii) experience using UML (*YearsUML*) to analyze such effects.

ExpModeling and *ExpServ* was collected by using pre-experiment questionnaire with open and 5-point Likert scale questions. *ExpModeling* and *ExpServ* were defined as YES/NO factors, calculated as the arithmetic mean among responses to questions related to each factor. We considered that a subject has experience if his/her calculated arithmetic mean value is greater or equal to the 2.5. The number of years using UML (open question) was used to calculate the factor *YearsUML* (*i.e.*, the YES/NO value was calculated through the Boolean expression *YearsUML*>=3).

5.2 Experiment Preparation and Execution

The experiment was planned to be conducted on two sessions. On the first day, a training session of 150 minutes was performed before the experimental session, the goal of this training session was to present the topics described in Table 2. The execution of the experimental took place on the second day. The experimental session had an expected duration of 45 minutes, however the subjects were allowed to finish the experiment even when the 45 minutes was over in order to mitigate the possible ceiling effect [31]. The experimental session consisted on two tasks, whose details are summarized in Table 2.

Multiple documents were designed as instrumentation for the quasi-experiment (available for download at [22]). The documentation of the experimental tasks included: i) a booklet that contains the description of the *Reservation System* integration scenario and the tasks to be performed by the subjects; ii) an annex with the Reservation System's *Current*

Architecture Model; iii) an annex with the *Increment Integration Specification Guidelines*; iv) a *Response Sheet* that contains the *IAM* after applying the *DIARy-specification-profile*, which subjects had to complete as they specify the increment integration by executing the Task 1; and v) the *Cloud Artifacts Model Editor* prototype which was used by subjects to update the *Increment Cloud Artifacts Model* and to generate reconfiguration scripts in Task 2. The training material consisted of a set of slides containing the description of the DIARy method and the use of the *DIARy-specification-profile*. Before the training and the experimental session, we executed a pilot study with an expert in software architectures and his observations were taken into account to improve both the experiment design and the experimental material.

Table 2. Schedule of the Quasi-Experiment

Session	***Task***	
Training session (150 min)	Basis on software architectures and architectural reconfiguration methods	
	Notations to describe service architectures and SoaML	
	DIARy method overview	
	Training in the DIARy method activities	
Experiment session (45 min)	Demographic & Background pre-experiment questionnaire	
	Task 1: Specify Increment Integration	
		Task 1.1: Identification of the architectural elements of the current architecture that will be affected by integration.
		Task 1.2: Specification of the architectural changes that integration will produce over the current architecture. For each element in the IAM subjects specified how will affect the Current Architecture Model.
		Task 1.3: Specification of how the changes on services workload is expected to be managed
	Task 2: Reconfigure Architecture	
		Task 2.1: Update models with information about the endpoints used by orchestration services to expose their operations. The participants accessed to the Microsoft Azure Management Portal to obtain information of deployment of the new Reservation System's orchestration service and updated the corresponding configuration elements (Exposed EndPoints) of the Cloud Artifact Model
		Task 2.2: Update the models with information about the endpoints used by services to invoke operations of their related services. The participants accessed to the Microsoft Azure Management Portal to obtain information of deployment of the service that initiate the interaction and updated the corresponding configuration elements (Invoked Endpoints) of the Cloud Artifact Model.
		Task 2.3: Generate the dynamic reconfiguration scripts and verification of the correctness of the reconfiguration script generated based on the increment integration specification (Task 1) and deployment information (Task 2.2 and Task 2.3).
		Task 2.4: Modify the architecture of the cloud application in the Windows Azure cloud environment by using the reconfiguration script. Reconfigure the actual running Reservation System`s architecture by changing links among services.
	Evaluation Questionnaire.	

The quasi-experiment was held with one group of twenty computer science students: 9 PhD-level students with profile PRO1, 6 PhD-level students with profile PRO2, 2 master-level students with profile PRO3, and 3 undergraduate-level students with profile PRO4. The quasi-experiment took place in a single room, and no interaction between subjects was allowed. The conductors of the experiment clarified the questions and doubts of the subjects that arose during the experimental session.

Data for this quasi-experiment were collected during the execution of the DIARy method activities by using the *Response Sheet* (for Task 1) and the *Cloud Artifacts Model*

Editor as well as the Microsoft Azure© cloud environment (for Task 2). Each participant was handling the reconfiguration a different service instance, in order to avoid cross effects due to the modifications of the service. The participants used the booklet to log the start and finish time of each task.

5.3 Data Analysis

The data analysis was performed by using the SPSS v.20 statistical tool for Windows with an $\alpha = 0.05$. In this analysis, we used descriptive statistics and statistical tests to analyze the collected data.

First, we performed an analysis of the responses to the pre-experiment questionnaire concerning to participants' background (knowledge and experience in modeling and web service development). Almost all subjects (90% of the subjects) possessed knowledge about UML and had some basic knowledge about service-oriented architecture modeling. With regard to knowledge in software development by applying cloud related development approaches, 75% of the subjects had developed web services and just 13% of them had deployed web services on cloud environments.

5.3.1 Qualitative Analysis

In order to analyze the effect that *ExpModeling, ExpServ, and YearsUML* may have on dependent variables, we performed the non-parametric Mann-Whitney test since these three variables (i.e., PEOU, PU and ITU) are measured by using aggregation of ordinal data (i.e., the average of various Likert scales). Table 3 shows the results of this test, which allowed us to verify that the *ExpModeling, ExpServ, and YearsUML* factors had no statistically significant effects on the subjective variables under study.

Table 3. *ExpModeling, ExpServ, and YearsUML* effect test results

	Factor		
Variable	***ExpModeling***	***ExpServ***	***YearsUML***
PEOU	0.238	0.358	0.261
PU	0.624	0.440	0.131
ITU	0.238	0.358	0.080

Table 4 shows a summary of the overall results for the subjective variables per factor.

Table 4. Descriptive Results per *ExpModeling, ExpServ, and YearsUML*

		Variable					
		PEOU		**PU**		**ITU**	
Factor	***Level***	$\tilde{X}$	σ^2	$\tilde{X}$	σ^2	$\tilde{X}$	σ^2
ExpModeling	No Exp.	4.000	0.578	4.357	0.328	4.667	0.717
	Exp. in UML	4.400	0.297	4.357	0.218	4.667	0.183
ExpServ	No Exp.	4.200	0.487	4.429	0.296	4.667	0.604
	Exp. in Serv.	3.400	0.520	3.857	0.048	4.000	0.148
YearsUML	<3	4.000	0.556	4.286	0.337	4.333	0.735
	>=3	4.400	0.378	4.429	0.158	4.667	0.151

We used the median and variance deviations as descriptive statistics for qualitative subjective variables PEOU, PU, and ITU. It can be noticed that all the variables are on average bigger than the Likert's scale neutral value equals to 3, meaning that, under the experimental conditions, the method is perceived as easy to use and useful and that the subjects show certain intention to use *DIARy* in the future.

It can be also noted that for the *ExpModeling* and *YearsUML* factor the values of each variable (PEOU, PU, ITU) are better for the more experienced participants, whereas for the *ExpServ* factor, we observe better values for the less experienced participants.

We checked the statistical significance of the results by performing the one-tailed one-sample Wilcoxon test with a test value equal to three for each group (see Table 5). For every variable, the results were found to be statistically significant in each sub-group, except for the participants with experience in service development. This is probably owing to the fact that the number of subjects in this group is low (i.e. N=3). We executed the one-tailed one-sample Wilcoxon test for the whole set of participants (N=20) in order to verify whether the data can be considered significant since there were no statistical differences among populations. The results of the test were p-value=0.000 for the three variables, then these results can be considered to be statistically significant for the population as a whole.

Table 5. One-Tailed One-Sample Wilcoxon Test Results

Variable	***ExpModeling***		***ExpServ***		***YearsUML***	
	No Exp.N=12	**Exp N=8**	**No Exp.N=17**	**Exp.N=3**	**<3N=11**	**>=3N=9**
PEOU	0.011	0.011	0.001	0.180*	0.014	0.012
PU	0.003	0.012	0.000	0.109*	0.004	0.008
ITU	0.004	0.011	0.001	0.102*	0.008	0.007

* Not statistically significant (p-value > 0.05)

The analysis of the open-questions' responses revealed that the majority of the participants highlighted the reduction of the reconfiguration technical depth as reasons for the adoption of *DIARy* in the future. However important issues which will allow us to improve the DIARy method were exposed. Participants pointed the lack of automated support in certain tasks. They recommended that the *Cloud Artifacts Model Editor* should be integrated with cloud environments to collect the deployment information, and also the need of an automated identification of architectural elements that already exist in the cloud application to analyze the architectural impact.

5.3.2 Quantitative Analysis

Similar to the process carried out for analyzing the qualitative variables, we first analyzed the effects that *ExpModeling*, *ExpServ* or *YearsUML* may have on the quantitative variables. We applied the Shapiro-Wilk test to check whether the data was normally distributed in order to select the statistical tests to be applied since the sample size was smaller than 50. Effectiveness and Efficiency approaches a normal distribution (Shapiro-Wilk $p-value \geq 0.05$) for each combination of groups and then the effect of *ExpModeling, ExpServ, and YearsUML* can be analyzed using the three-way ANOVA test. We carried out a three-way ANOVA test including *ExpModeling, ExpServ, and YearsUML* as blocking variables to verify that there were no significant effects on the Effectiveness and Efficiency variables (*p-values* ≥ 0.05). Table 6 summarizes the results for each variable by factor. It can be observed that, on average, the participants with modeling experience (*ExpModeling* and *YearsUML*>=3 years) obtained better results in terms of effectiveness and efficiency. Effectiveness for participants with *ExpModeling* and *YearsUML* was 84% and 85% respectively (measured by the correctness of the subject responses).

Regarding the experience on service development (i.e., *ExpServ*) it can be observed that in average the no experienced participants obtained better results on terms of effectiveness and efficiency than the experienced participants, although these differences were not found to be statistically significant. This can be owing to the fact that the subjects work with models at a high abstraction level which does not require high-level technical skills to execute experimental tasks.

Table 6. Descriptive Statistics of the Quantitative Variables

		Variable			
		Effectiveness		***Efficiency***	
Factor	***Level***	$\bar{x}$	σ	$\bar{x}$	σ
ExpModeling	No Exp.	0.770	0.185	0.032	0.017
	ExpModeling	0,839	0.171	0.035	0.013
ExpServ	No Exp.	0.818	0.175	0.035	0.016
	ExpServ.	0.683	0.180	0.023	0.001
YearsUML	<3	0.758	0.193	0.036	0.019
	>=3	0.847	0.154	0.030	0.010

5.4 Threats to the Validity

In this section, we analyze the main issues that may threatened the validity of the quasi-experiment, by considering the four types of threats proposed in [32]:

Internal Validity: The main threats to the internal validity were: persistence effects, participants' experience and the understandability of the documents. In order to avoid persistence effects, not only the cloud application used as part of integration scenario and the one used as example in the training session belong to different domains but also the architectural changes they suffered due to integration of new services were different. The participants' experience threat was mitigated by defining a Pre-experiment questionnaire to analyze the possible differences between experienced and non-experienced subjects. This allowed us to reject the possible impact that the factor *background* may have on the variables under study. Even tough participants were students at the moment of the experiment, under certain conditions, there is no great difference between students and professionals [33], and they could be considered as the next generation of professionals [34]; therefore we believe that their ability to understand architectural models, service development principles, and to evaluate dynamic architecture reconfiguration processes can be comparable to that of typical novice practitioners. Finally, the understandability of the material was minimized by clearing up all of the misunderstandings that appeared in the experimental session; the materials were also reviewed by a research group member (not part of experimenters) but also by an expert in software architectures who did the pilot study prior to the experimental sessions.

External Validity: The main threat to external validity is the representativeness of the results which might be affected by the design of the evaluation, the size and complexity of the tasks, and the participant context selected. To alleviate this threat, we included in the experiment only the most representative DIARy method tasks applied during the deployment of new cloud services. The size and complexity of the tasks might also have affected the external validity. We attempted to propose a set of experimental tasks with a sufficient level of complexity, given the time constraints of the sessions. We have also provided tool prototypes in order to facilitate the execution of tasks; however, to completely address this issue we should integrate the provided tools fully integrated with the cloud environment management system. With regard to the participants' experience, the quasi-experiment was conducted with students with certain knowledge about system modeling and web services development. The preliminary results obtained could be only considered to be representative for a model-driven development environment and with populations composed of novice practitioners. However, as further work, we intend to conduct more experiments involving to professionals, bigger groups and greater homogeneity among them.

Construct Validity: The main threat to construct validity is to reflect how the metrics that have been studied represent the goals of the researcher. The subjective variables are based on the Technology Acceptance Method (TAM), which is a well-known and empirically validated model for the evaluation of information technologies [27]. Thus, the main threat is the reliability of the evaluation questionnaire. We carried out a Cronbach's alpha test for each set of questions related to each subjective variable. The results of the Cronbach reliability test were greater than the minimum acceptance threshold $\alpha > 0.70$ [35] (i.e., PEOU_ $\alpha = 0.844$, PU_ $\alpha = 0.781$, ITU_ $\alpha = 0.747$).

Conclusion Validity: The main threats to the conclusion validity is the validity of the statistical tests applied. We reduced this threat by applying a set of commonly accepted tests that are employed in the empirical SE community [35].

6 CONCLUSIONS AND FUTURE WORK

The reconfiguration of cloud service architectures triggered by the integration of new services in cloud environments introduces a number of challenges. In particular, the high-availability required for cloud applications implies that this reconfiguration should be done while the application is in execution. In this paper, we presented i) an improvement of the DIARy method that allows developers to specify new services as software increments, and ii) the tool support to generate the implementation code for the services integration logic and the deployment and reconfiguration scripts specific to the cloud environment in which the service will be deployed. We also introduced the results of a quasi-experiment aimed at evaluating the *perceived ease of use*, *perceived usefulness*, and *intention to use* of a group of participants that applied the DIARy method, as well as the participants' performance. The main results indicated that under the experimental conditions the DIARy method is perceived to be ease of use, useful, and likely to be used, independently of the participants' background. Participants with experience in modeling and those with less experience in service development perceived the method as easier to use and useful, and they showed more intention to use it in the future. To the best of our knowledge, this is the first empirical study that provides evidence of the usefulness of the dynamic reconfiguration process for supporting the incremental implementation and integration of software increments into existing cloud applications.

We also identified some limitations. Architectural reconfiguration is achieved by deploying/redeploying services and by updating dependencies among them; however, it is not always feasible to redeploy a service when it has running instances. This is a challenging task, especially if those instances are running of different cloud providers since they offer different mechanisms to manage instances. The instance management of the proposed method is in initial stage and does not support the management of instances of services running on different cloud environments. Additionally, some specific resource characteristics offered by a cloud provider might not be used efficiently due to some resource characteristics are abstracted in our models, making unable to take advantage of some proprietary advanced characteristics. Fortunately, the model-driven approach followed by our method enables us to abstract the instance management mechanisms, as well as to describe some proprietary advanced characteristics at a detailed level. In addition, our method may have impact in both research and industry because it provides mechanisms to deal with the integration impact of cloud architectures at a high abstraction level and tools to automate the process. Currently, we are investigating how to adopt the DevOps paradigm. This paradigm will allow us to move from an incremental integration to a continuous integration approach joining development and operations activities, as well as, to define a comprehensive view of the tool chaining needed. We are also exploring the impact of other architectural styles, such as microservices. As future work, we plan to study and implement the transformation code to support other cloud environments, such as Google App Engine or Amazon EC2. We also plan to better integrate and package our different Eclipse plugins into a standalone application. Regarding the empirical validation, we plan to conduct replications of the quasi-experiment by considering homogenous groups with a larger number of subjects and different experimental objects in order to improve the representativeness of our results. We also plan to conduct other empirical studies, such as industrial case studies, to gather empirical evidence from practitioners of the actual feasibility of the approach.

ACKNOWLEDGEMENTS

This research is supported by the Value@Cloud project (MINECO TIN2013-46300-R), DIUC_XIV_2016_038 project, and the Microsoft Azure Research Award.

REFERENCES

1. Lane, S., Gu, Q., Lago, P., Richardson, I.: A Pragmatic Approach for Analisys and Design of Service Inventories. Service Oriented Computing and Applicat. pp. 1–19 (2013).
2. Motta, G., Sfondrini, N., Sacco, D.: Cloud Computing: An Architectural and Technological Overview, International Joint Conference on Service Sciences (IJCSS). pp. 23–27. IEEE, Shanghai (2012).
3. Chhabra, B., Verma, D., Taneja, B.: Software Engineering Issues from the Cloud Application Perspective. Int. J. Inf. Technol. Knowl. Manag. 2, 669–673 (2010).
4. Breivold, H.P., Crnkovic, I., Radosevic, I., Balatinac, I.: Architecting for the Cloud: A Systematic Review. 17th International Conference on Computational Science and Engineering. pp. 312–318. IEEE (2014).
5. Zuñiga-Prieto, M., Abrahao, S., Insfran, E.: An Incremental and Model Driven Approach for the Dynamic Reconfiguration of Cloud Application Architectures. 24th International Conference on Information Systems Development (ISD) Harbin, China, (2015).
6. Zuñiga-Prieto, M., Gonzalez-Huerta, J., Abrahao, S., Insfran, E.: Towards a Model-Driven Dynamic Architecture Reconfiguration Process for Cloud Services Integration. 8th International Workshop on Models and Evolution (ME@ MoDELS). pp. 52–61 (2014).
7. Guillén, J., Miranda, J., Murillo, J.M., Canal, C.: Developing Migratable Multicloud Applications based on MDE and Adaptation Techniques. Proceedings of the Second Nordic Symposium on Cloud Computing & Internet Technologies. 30–37 (2013).
8. Frey, S., Hasselbring, W.: The CloudMIG Approach: Model-Based Migration of Software Systems to Cloud-Optimized Applications. International Journal on Advances in Software,. 4, 342–353 (2011).
9. Ardagna, D., Di Nitto, E., Casale, G., Petcu, D., Mohagheghi, P., Mosser, S., Matthews, P., Gericke, A., Ballagny, C., D'Andria, F., Others: MODACLOUDS : A Model-Driven Approach for the Design and Execution of Applications on Multiple Clouds. Proceedings of the 4th International Workshop on Modeling in Software Engineering. pp. 50–56 (2012).
10. Bergmayr, A., Troya, J., Neubauer, P., Wimmer, M., Kappel, G.: UML-based Cloud Application Modeling with Libraries, Profiles, and Templates. In CloudMDE@MoDELS. pp. 56–65 (2014).

11. Guillén, J., Miranda, J., Murillo, J.M., Canal, C.: A UML Profile for Modeling Multicloud Applications. European Conference on Service-Oriented and Cloud Computing. pp. 180–187 (2013).
12. Brandtzæg, E., Mosser, S., Mohagheghi, P.: Towards CloudML, a Model-based Approach to Provision Resources in the Clouds. 8th European Conference on Modelling Foundations and Applications (ECMFA). pp. 18–27 (2012).
13. Chef Software, I.: DevOps and the Cloud: Chef and Microsoft Azure (2015).
14. Meireles, F., Malheiro, B.: Integrated Management of IaaS Resources. European Conference on Parallel Processing. pp. 73–84. Springer International Publishing (2014).
15. Newman, S.: Building Microservices. O'Reilly Media, Inc. (2015).
16. Brogi, A., Ibrahim, A., Soldani, J., Carrasco, J., Cubo, J., Pimentel, E., D'Andria, F.: SeaClouds: A European Project on Seamless Management of Multi-Cloud Applications. ACM SIGSOFT Softw. Eng. Notes. 39, 1–4 (2014).
17. Jamshidi, P., Ghafari, M., Ahmad, A., Pahl, C.: A Framework for Classifying and Comparing Architecture-Centric Software Evolution Research. 17th European Conference on Software Maintenance and Reengineering. pp. 305–314. IEEE, Genova (2013).
18. OMG: Service Oriented Architecture Modeling Language (SoaML)-Specification for the UML Profile and Metamodel for Services (UPMS), (2008).
19. Zuñiga-Prieto, M., Insfran, E., Abrahao, S.: Architecture Description Language for Incremental Integration of Cloud Services Architectures. IEEE 10th Symposium on the Maintenance and Evolution of Service-Oriented Systems and Cloud-Based Environments (MESOCA), Raleigh, USA (2016).
20. Costa, B., Pires, P.F., Delicato, F.C., Merson, P.: Evaluating REST Architectures-Approach, Tooling and Guidelines. J. Syst. Softw. 112, 156–180 (2014).
21. Humble, J., Farley, D.: Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Pearson Education (2010).
22. Zuñiga-Prieto, M.: DIARy-Method: Experimental Material, http://thediarymethod.azurewebsites.net/.
23. Shadish, W.R.., Cook, T.D., Campbell, D.T.: Experimental and Quasi-Experimental Designs for Generalized Causal Inference. Houghton Mifflin Company, Boston, ME, USA (2002).
24. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering. Springer Heidelberg (2012).
25. Basili, V.R., Caldiera, G., Rombach, H.D.: The Goal Question Metric Approach. Encyclopedia of Software Engineering. pp. 1–10. Wiley (1994).
26. SOFTEAM R&D web-site: Full SoaML Tutorial - BPMN, SoaML, BPEL Transformation, http://rd.softeam.com/demos/soaml/.
27. Davis, F.F.D.: Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. MIS Q. 13, 319–340 (1989).
28. Venkatesh, V.: Determinants of Perceived Ease of Use : Integrating Control , Intrinsic Motivation , and Emotion into the Technology Acceptance Model. Inf. Syst. Res. 11, 342–365 (2000).
29. Moody, D.L.: Dealing with Complexity: a Practical Method for Representing Large Entity Relationship Models, (PhD Thesis), University of Melbourne (2001).
30. Hu, P.J., Chau, P.Y.K., Liu Sheng, O.R., Tam, K.Y.: Examining the Technology Acceptance Model Using Physician Acceptance of Telemedicine Technology. J. Manag. Inf. Syst. 16, 91–112 (1999).
31. Sjøberg, D., Anda, B., Arisholm, E., Dyba, T., Jørgensen, M., Karahasanovic, A., Vakác, M.: Challenges and Recommendations When Increasing the Realism of Controlled Software Engineering Experiments. In: Empirical Methods and Studies in Software Engineering. pp. 24–38 (2003).
32. Cook, T., Campbell, D.: Quasi-experimentation: Design & Analysis Issues for Field Settings. Houghton Mifflin Company (1979).
33. Höst, M., Regnell, B., Wohlin, C.: Using Students as Subjects—A Comparative Study of Students and Professionals in Lead-Time Impact Assessment. Empir. Softw. Eng. 5, 201–214 (2000).
34. Kitchenham, B.A., Pfleeger S.L., Pickard L.M., Jones P.W., Hoaglin D.C., El Eman K., Rosenberg J.: Preliminary Guidelines for Empirical Research in Software Engineering. Softw. Eng. IEEE Trans. 28, 721–734 (2002).
35. Maxwell, K.: Applied Statistics for Software Managers. Prentice Hall (2002).