An Incremental and Model Driven Approach for the Dynamic Reconfiguration of Cloud Application Architectures

Miguel Zúñiga-Prieto *mzuniga@dsic.upv.es*
Universitat Politècnica de València
Valencia, Spain

Silvia Abrahao *sabrahao@dsic.upv.es*
Universitat Politècnica de València
Valencia, Spain

Emilio Insfran *einsfran@dsic.upv.es*
Universitat Politècnica de València
Valencia, Spain

Abstract

In incremental development approaches, the integration of new services into the actual cloud application may trigger the dynamic reconfiguration of the cloud application architecture, thus changing its structure and behavior at runtime. This paper presents a model driven approach that uses the specification of how the integration of new services will change the current cloud application architecture to obtain: i) the orchestration of services, ii) skeletons of interface implementations, and iii) the operationalization of reconfiguration actions to be applied at runtime. This approach follows the DIARy-process, which defines the activities needed to reconfigure dynamically the architecture of cloud services. The feasibility of the approach is illustrated by means of a case study that uses Microsoft Azure© as a service deployment platform. WCF Workflow services are generated and deployed for orchestration, whereas XML transformation files are generated to update services' binding configurations at runtime.
Keywords: software architecture, model driven development, dynamic reconfiguration, cloud computing, SoaML.

1. Introduction

Cloud applications are distributed software systems composed of services (typically Web services) that run on top of third-party cloud platforms, consume cloud platform resources (e.g., execution environment, storage, virtual machines, message queue services), and follow a pay-per-use pricing model.

The development of a cloud application differs from that of traditional software systems since, in general terms: i) services are developed using an incremental/iterative development process; ii) services are highly reusable; and iii) system models with a holistic definition of all functionalities to be produced do not necessarily exist [16]. Additionally, in an incremental development approach, the integration of software increments (which include a set of services) may trigger the dynamic reconfiguration of the actual cloud application architecture. A cloud application architecture can be defined as the fundamental organization of a system, embodied in its cloud services, their relationships to each other and the environment, and the principles governing its design and evolution (adapted from [13]). The dynamic reconfiguration of cloud application architectures creates and destroys instances of architectural elements and changes their relationships at runtime, without stopping the system. Also, it is important in cloud environments to manage the service instances on different platforms. Successful cloud adoption thus requires guidance around planning and integrating cloud application increments.

Cloud services must be highly available, signifying that some challenges must be solved in order to support dynamic reconfiguration and minimize system disruptions during integration. The architectural changes expected to occur during the integration should be planned beforehand when modeling integration. Furthermore, implementation should also comply with service-oriented principles. This is not an easy task since cloud development platforms provide various alternatives with which to both organize source code and use their services. In addition, cloud services are built according to cloud provider technology (e.g., programming language, protocols, APIs usage), which very often leads to a tight coupling among them. Mechanisms that facilitate the making of implementation decisions independently of the cloud platform are therefore required.

We believe that Model-Driven Development (MDD) provides a good support for the dynamic reconfiguration of cloud application architectures. It allows us to capture technology-independent application information, make design artifacts reusable, change reconfiguration specifications and automate the cloud application architecture reconfiguration process. The Service oriented architecture Modeling Language (SoaML) [3] which is an OMG standard specifically designed for the modeling of service-oriented architectures, facilitates service modeling and design activities while following an MDD approach. However, service-oriented design concepts are not fully understood at implementation level and service-based systems are often developed without taking into account good software engineering techniques [21].

To tackle these problems, in this paper we present a model-driven approach to support the dynamic reconfiguration of cloud application architectures caused by the integration of cloud service increments. We extend the expressiveness of SoaML, providing it with features that allow us to specify how increment architectures will be integrated into the current cloud application architecture. This specification facilitates the generation of cloud artifacts (software artifacts to be used in cloud environments) that assist in the processes of both increment integration and the dynamic reconfiguration of the current cloud application architecture. We have followed the DIARy-process [23], which defines activities for the dynamic architecture reconfiguration of cloud services.

The remainder of this paper is structured as follows. Section 2 discusses related works and identify the reconfiguration needs. Section 3 presents an overview of the DIARy-process. Section 4 describes how our reconfiguration approach helps software architects to plan the reconfiguration actions that will occur when deploying a cloud service increment, while Section 5 presents the use of our approach to generate software artifacts that facilitate the dynamic reconfiguration of cloud application architectures. Section 6 illustrates the use of our approach in a case study. Finally, Section 7 presents our conclusions and future work.

2. Related Work

Breivold et al. [5] 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.

Over the last years, several proposals for the dynamic reconfiguration of software architectures have been proposed. Despite the fact that it takes place at runtime, the way in which an application is designed and implemented helps achieve this reconfiguration. In this section, we analyze how researchers and practitioners support the modeling of cloud applications and the dynamic reconfiguration of cloud software architectures.

We highlight three approaches that propose means to document design decisions in cloud environments: CAML [2], a UML profile that enables cloud deployment topologies to be wired with cloud providers' specific offerings; MULTICLAPP [11], a framework for the development of cloud applications, which includes a UML profile in order to model cloud applications as a composition of software artifacts that can be deployed across multiple clouds; and CloudML [4], which proposes a cloud modeling language for describing the resources that a given application may require from existing clouds. Although these proposals provide information for multi-cloud implementation and provisioning or deployment, they lack mechanisms to specify

the impact of integrating increments into the current cloud application architecture. This capability is relevant as cloud services are developed following an incremental process. In addition, when modeling increment integration, software architects should be able to work with independent models in order to describe increments, rather than being constrained to modify the overall system model [17], and to update it by including the services to be integrated in it.

With regard to proposals for dynamic reconfiguration, the ongoing SeaClouds [6] project proposes a platform which performs a 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 [1] project is, meanwhile, ongoing research into 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 do not allow the specification of the architectural changes produced during reconfiguration. These proposals take into account alternatives as regards provisioning and deployment; however, they do not take into account implementation alternatives that influence the obtaining of loosely coupled services, which facilitate scalability and the re-deployment of services in different clouds.

Finally, in the aforementioned proposals, the reconfiguration/re-deployment starts as the result of monitoring activities, and changes are therefore made to improve quality attributes; however, adaptive changes (e.g., software increments resulting from new functionalities) that require architectural changes are not taken into account.

3. The DIARy-process

The integration of an increment into the current cloud application not only provides new functionalities for clients, but also functionality updates and the repairing of defects. From an architectural point of view, when an increment is integrated into the current cloud application its architectural elements update or are incorporated into the current cloud application architecture, thus reconfiguring it. The DIARy-process (see Fig. 1) proposes activities whose aim is to facilitate the architectural reconfiguration caused by increment integration; its main activities are:

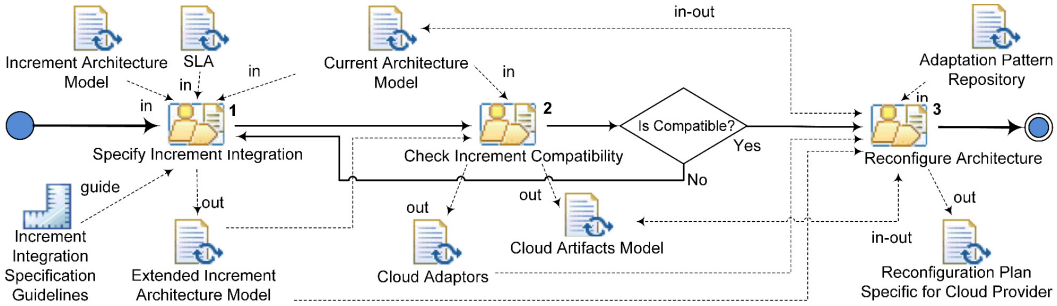

Fig. 1. The DIARy-process

1. *Specify Increment Integration:* Software architects take the architectural description of an increment *(Increment Architecture Model)* as input and specify how each of its architectural elements will change the current cloud application architecture (*Current Architecture Model*). They perform this activity by following *Increment Specification Guidelines* and making design decisions based on *SLA* terms; they then document their decisions in an *Extended Increment Architecture Model*. This model includes: i) information regarding the structure and behavior of the cloud services delivered in an increment; ii) information concerning how the increment's architectural elements collaborate to reconfigure the current cloud application architecture, which describes the integration impact; and iii) information regarding services that can be used to take advantage of cloud environment properties.
2. *Check Increment Compatibility:* Software architects participate in verifying whether the increment's architecture is compatible with the current cloud application architecture. If

discrepancies exist between the interfaces of these architectures (e.g., different names for methods and services, different message ordering), they apply model-to-text transformations that generate skeletons of cloud artifacts that are specific to a cloud platform and that solve discrepancies (*Cloud Adaptors*). This activity requires the use of adaptation techniques. Several other authors propose adaptation techniques that can be used, such as [7].

3. *Reconfigure Architecture:* This activity takes the following as inputs: the *Extended Increment Architecture Model*, which documents the specification of increment integration*;* the *Cloud Artifacts Model*, which is a high-level description of how cloud artifacts should be organized and which cloud platform services should be used in order to implement increment specifications; and *Adaptation Patterns* [10], which represent generic and repeatable solutions that can be used to manage architectural changes in recurring architectural adaptation problems. Cloud developers use the inputs for this activity to apply model transformations that generate cloud artifacts which both implement architectural elements and operationalize the reconfiguration actions needed to reconfigure the current cloud application architecture at runtime (*Reconfiguration Plan Specific for Cloud Provider*).

For more information about the DIARy process refer to [23]. The contribution of this work is to define the *specification of the increment integration* and the actual *reconfiguration of the architecture.* These activities are explained in the following sections.

4. Specifying Increment Integration

We provide a modeling solution with which to specify how the increment's architecture will be integrated into the current cloud application architecture. This modeling solution provides:

- *Support for the Partial Specification of Software Architectures:* Cloud services are developed using an iterative and incremental process, in which fragments of software systems are developed at different rates or times, and then integrated. Our proposal does not force software architects to modify the overall system architecture model in order to include the architectural elements of the increment in it. By avoiding this, we help them track changes and differentiate increment architectural elements from current architecture elements.
- *Support for the Specification of Integration Impact:* Our proposal allows software architects to specify the way in which each element of the increment's architecture impacts on the current cloud application architecture (plan ahead static changes).
- *Support for the Design of Services Deployed on Cloud Environments:* Decisions made during early phases of a development process influence later implementation/provisioning/deployment decisions. One of the most important properties of cloud environments is their capacity to provide resources on demand. Our proposal allows software architects to provide information about the workload expected in a service, thus helping developers working in later phases to make decisions that take advantage of cloud platform resources.

Architectural descriptions that use the Unified Modeling Language (UML) have gained popularity and have been widely adopted in industry [17]. Initiatives such as [8], [15] extend UML capabilities in order to model service architectures, the Service oriented architecture Modeling Language (SoaML) [3] (an OMG specification) being the most widely adopted [22]. We have extended both UML2.4 and SoaML to take advantage of already existing modeling languages, rather than starting from scratch by defining a new one. The use of UML profiles to extend these languages will allow us: i) to specify the architectural impact of the integration from a high-level point of view; and ii) to describe the orchestration (interaction coordination) of services, separating the functional logic of cloud services from the integration logic, which will help attain the architectural dynamic reconfiguration.

4.1. The DIARy-specification-profile

The profile defined in this section (see Fig. 2) extends both UML2.4 and the SoaML profile. The main SoaML elements to be extended are:

- *Participant:* Plays the role of service provider, consumer, or both.

- *Services Architecture:* Defines how participants work together to provide and use services which are described as Service Contracts in a Services Architecture.
- *Service Contract:* Represents an agreement between the participants involved regarding how the service is supposed to be provided and consumed. Service contracts are frequently part of one or more Services Architectures.
- *Collaboration Use:* Indicates explicitly how a Collaboration (Service Contract or Services Architecture) is fulfilled by the different inner parts.

In order to avoid the situation of software architects having to work with the overall cloud application architecture model when specifying increment integration, we extend the UML Collaboration element by using the stereotype «*ExtendedIncrementArchitecture*». Its semantics is similar to that of the SoaML Services Architecture, but it also describes how its inner parts collaborate to reconfigure the current cloud application architecture.

In order to support the specification of the impact of the integration of increments, we extend elements subject to change («*ParticipantUse*», «*ServiceContractUse*», and «*RoleBinding*») by including the tagged-value *architecturalImpact* in them, whose possible values are: *Add*, *Modify*, or *Delete*, which denote architectural changes; and *Reference*, which denotes elements of the current cloud application architecture that will not change but will interact with the increment's architectural elements. By using the tagged-value *representCloudArtifact* we allow software architects to specify whether architectural changes need to be propagated to their related cloud artifacts.

When dealing with workload changes in elastic scenarios, the use of Cloud Application Management Patterns [19] suggests managing performance by adding or removing cloud platform resources. However, in less elastic scenarios in which delaying the processing of requests might be more effective than processing them immediately, the use of message queues is suggested. We support both scenarios by extending the SoaML's notation element «*CollaborationUse*» with the stereotype «*ServiceContractUse*»**,** and including the tagged-values *elasticityLevel* and *delayLevel* in it, whose possible values are: *None*, *Low*, *Medium*, and *High*. The assigned value will be used by cloud developers in later development phases to select implementation/provisioning/deployment alternatives, or cloud platform services that satisfy SLA terms or other requirements.

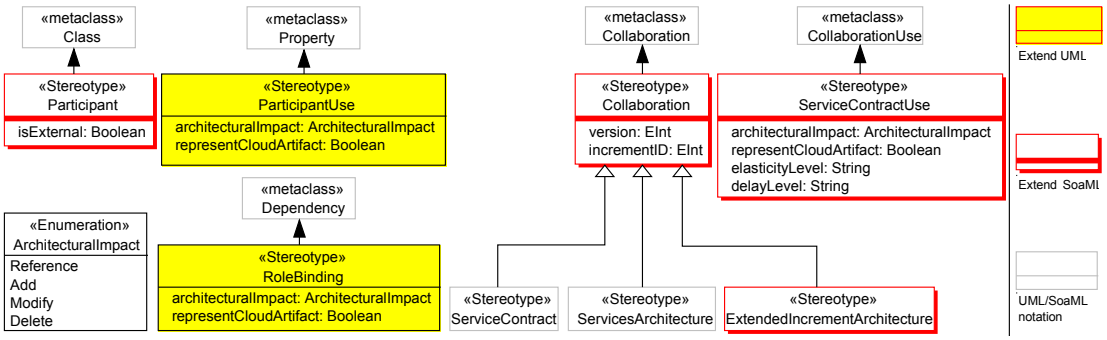

Fig. 2. The DIARy-specification-profile

Implementation decisions may vary depending on whether services will be provided or consumed by external participants, and we therefore extend the SoaML's notation element «*Participant*» by adding the tagged value *isExternal*. Finally, in order to facilitate version management, and preserve the order of the evolution process and the coherence of interactions among instances [20], we extend the *Collaboration* element by including the tagged-values *version* and *incrementID*.

5. Reconfiguring Cloud Application Architectures

In order to support the *Reconfigure Architecture* activity of the DIARy-process (see Fig. 1), in this section we delineate the *Cloud Artifacts Model*'s metamodel and how it is used to support this activity. This model represents concepts that are independent of cloud platform which are

used to describe and organize the cloud artifacts needed: to implement *Extended Increment Architecture Model* elements, to invoke cloud platform services, and to provision/deploy cloud platform resources in accordance with the decisions made during the development process.

5.1. The Cloud Artifact Model's Metamodel

The integration of increment functionalities into the current cloud services is achieved by allowing services to interoperate. Interoperability among the services deployed in cloud environments requires the orchestration of services [20]. The way in which cloud artifacts are organized and implement architectural designs simultaneously have an influence on the obtaining of loosely coupled services, which help support the interoperability and re-deployment of services on different cloud platforms. Our metamodel promotes the decoupling of the logic of the orchestration from the logic of participant's services, thus signifying that the orchestration of services can be offered as a reusable cloud service and facilitating the architectural dynamic reconfiguration by replacing either the orchestration or bindings to participants' cloud services.

The *Cloud Artifacts Model*'s metamodel (see Fig. 3) allows the representation of the cloud artifacts needed to deploy: a cloud service for each participant playing a role in a «*ServiceContractUse*» included in an *Extended Increment Architecture Model*; and a cloud service with which to orchestrate interactions among those cloud services. The correlation between the *Extended Increment Architecture Model* and the *Cloud Artifacts Model* is achieved by mapping architectural elements represented by *DIARy Architectural Element* classes with cloud artifacts represented by *Artifact* classes.

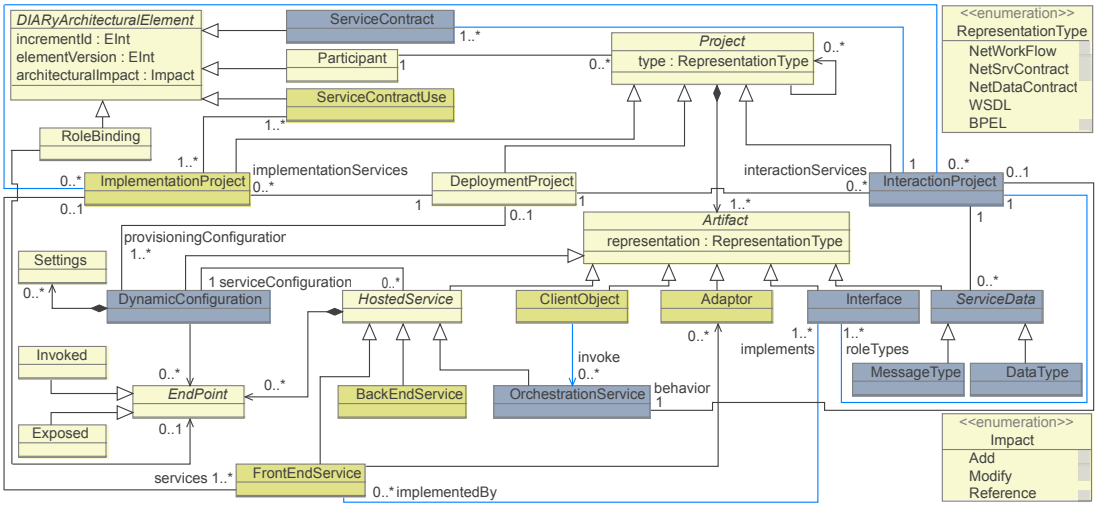

Fig. 3. Cloud Artifacts Model's metamodel

We use *Projects* to organize cloud artifacts and describe how their instances will use cloud platform resources. Instances of cloud services whose related cloud artifacts are included in the same *Interaction Project* or *Implementation Project* will share cloud platform resources and will be hosted in the same virtual machine; organizing cloud artifacts into exclusive or shared projects thus allows workload changes and running costs to be managed.

Interaction Projects are mapped onto *Service Contract* architectural elements and include cloud artifacts corresponding to cloud services that provide orchestration; cloud artifacts that: implement the interaction protocol (*Orchestration Service*); *Interfaces*, *Message Types* and *Data Types* (see Table 2).

Implementation Projects are mapped onto *Service Contract Uses* and include cloud artifacts corresponding to cloud services that expose the operation logic of service participants: *Front End Services* that implement interfaces defined in the corresponding *Service Contract*; *Back End Services* that use cloud platform services (e.g., message queues); *Client Objects* that initiate the service execution by invoking an *Orchestration Service*; or *Adaptors* that correct incompatibilities between interfaces. Finally, *Deployment Projects* facilitate the grouping of interac-

tion/implementation projects related to both a cloud application and a participant, which determine the cloud resources that must be provided in order to deploy the cloud services of which the participant cloud application is made up.

The *Dynamic Configuration* class is used to describe configuration *Settings* that could change at runtime, thus satisfying cloud design patterns for dynamic reconfiguration (e.g., [12], [9]). The aforementioned works suggest that the configuration settings should be stored outside the deployed service so that they can be updated without requiring the redeployment of entire package. This paper does not include an explanation of each class in the metamodel or its attributes, or the OCL validations needed to maintain a consistent model for reasons of space.

5.2. Cloud Artifacts Generation

In our approach, the *Reconfigure Architecture* activity uses the specification of the increment integration to generate cloud artifacts with which to support the implementation and deployment of services. During the implementation, it is necessary to *Generate Implementation Artifacts* defining model transformations that generate cloud artifacts that decouple the implementation of the orchestration's logic from the implementation of the services' logic. During the deployment, it is necessary to *Generate Reconfiguration Artifacts* defining model transformations that generate cloud artifacts that operationalize the reconfiguration actions.

Implementation: Generate Implementation Artifacts

In the first step, cloud developers execute model-to-model (M2M) transformations that deal with *alternative transformations* which organize cloud artifacts into projects in order to generate the *Cloud Artifacts Model*. *Alternative transformations* [14] take into account both the architectural impact and the requirements needed to manage the workload changes described during the increment integration specification (and documented in the *Extended Increment Architecture Model*). Table 1 shows examples of alternatives to organize cloud artifacts into projects applied when the *architecturalImpact* of a «*ServiceContractUse*» is *Add*.

Table 1. Alternatives to organize cloud artifacts into projects

Elasticity level	Delay level	Cloud artifacts organization
High	None	- An Orchestration Service in its own Interaction Project - One Front End Service per participant in its own Implementation Project
Low	High	- An Orchestration Service in a shared Interaction Project - One Front End Service per participant in a shared Implementation Project - A Back End Service to manage the delaying of requests (e.g., message queues) in the Implementation Project corresponding to the service provider

In the second step, cloud developers complete the *Cloud Artifacts Model* generated according to the cloud platforms on which the cloud services will be deployed by: i) providing provisioning and deployment information for *Deployment Projects* and configuration information for *Hosted Services*, creating or updating *Dynamic Configuration* or *Setting* classes; ii) specifying the *representation* of each cloud artifact (e.g., source code language). Once the model has been updated, cloud developers execute a model-to-text (M2T) transformation which generates cloud artifacts that are specific to a cloud platform, using the *Extended Increment Architecture Model* and the implementation/deployment/provisioning decisions described in the *Cloud Architecture Model* as input. Finally, cloud developers complete the cloud artifacts generated.

Deployment: Generate Reconfiguration Artifacts

In the first step, once the previously generated cloud artifacts have been deployed, cloud developers update the *Cloud Artifacts Model* with information regarding the EndPoints in which the corresponding cloud services are exposed. They then select the adaptation patterns best suited to integrating the increment's architecture into the current cloud application architecture from

the *Adaptation Pattern Repository*. In the following step, they execute M2T transformations in order to generate *Reconfiguration Plan Specific for Cloud Provider* cloud artifacts that operationalize the selected adaptation patterns according to: the increment integration specification, *Cloud Artifacts Model* and cloud platforms on which the services will be deployed. We consider patterns in which the integration: i) changes the implementation logic of an already deployed orchestration service; ii) requires the deployment of a new orchestration service in order to integrate new services with already existing ones; and iii) changes the implementation logic of an orchestration service and the services that participate in an interaction.

Finally, the *Extended Increment Architecture Model* is used as the input of the M2M transformations that update the *Current Architecture Model* according to the architectural impact described in the increment integration specification (represented in the input model).

6. Case Study

To illustrate the use of our approach, in this section we present an excerpt of a case study (adapted and extended from [3]) dealing with shipping of products among commercial partners. A manufacturing company wished to improve the technological support its dealers and partners were provided with. With this purpose, it considered building and deploying cloud services in an incremental manner. The initial version supported the fulfillment of dealers' orders and provide dealers with cloud services that would allow them to place and manage their orders, thus allowing a direct interaction between the customer's IT systems and the company's systems. The company then needed to incorporate a shipping process, and Increment-1 therefore provided the transport partner with cloud services to manage the orders to be shipped.

An explanation of how the DIARy-specification-profile and the model defined in our approach were used to support the activities of the DIARy-process is shown below.

6.1. Supporting the *Specify Increment Integration* Activity

The modeling tasks required in this activity were performed by using Papyrus, an Eclipse Modeling Framework (EMF) based modeling environment designed as an open source Eclipse component. In this activity, software architects take as input the *Increment Architecture Model* (see Fig. 4a) and the *Current Architecture Model* (see Fig. 4b), which were built by using the SoaML profile and the DIARy-specification-profile, respectively. The firs model describe the architecture of the Increment-1 by using a high-level representation of its services and their relationships (see «*ServicesArchitecture*» in Fig. 4a). It is a SoaML model and therefore includes the modeling of its internal components (e.g., interfaces that define a role type, interaction coordination – orchestration-, message types); these are not, however, shown in this paper for reasons of space. SoaML also provides various approaches with which to specify service architectures; we followed the Service Contract based approach because it is the most suitable when more than two parties are involved in a service [3]. The *Current Architecture Model* (see Fig. 4b) is, meanwhile, used to identify the architectural elements of the current architecture which, after integration, will change or will interoperate with architectural elements of the increment.

Integration of Increment-1 replaced the service *Place Order* (shown in bold type in Fig. 4b) with the service *Order With Shipping* (shown in bold type in Fig. 4a). The integration was modeled by applying the DIARy-specification-profile to the *Increment Architecture Model* (see Fig. 4a); the architectural element «*ServicesArchitecture*» Increment-1 was then tagged by using the stereotype «*ExtendedIncrementArchitecture*»; finally, its internal parts were tagged in order to specify how they change the current architecture (shown in bold type in Fig. 4c). For instance, the «*ServiceContractUse*» purchacse:OrderWithShipping and its related «*Role Binding*» were tagged with *architecturalImpact = Add*. Additionally, the «*ServiceContractUse*» purchase:PlaceOrder and its related «*RoleBinding*», which are not part of the «*ExtendedIncrementArchitecture*» element but will be replaced owing to integration, were included and tagged with *architecturalImpact = Delete* (shown in bold-italics in Fig. 4c). However, the manufacturer: Manufacturer «*ParticipantUse*» already existed in the «*ExtendedIncrementArchitecture*» element but its interface implementation needed to be updated to include the shipping logic,

and it was therefore tagged with *architecturalImpact = Modify*.

After specifying integration impact, the software architects specify how service's workload changes would be managed. The Order With Shipping service is expected to be a highly demanded service in which delays will not be allowed, and it was therefore tagged as *elasticityLevel=High* and *delayLevel=None*.

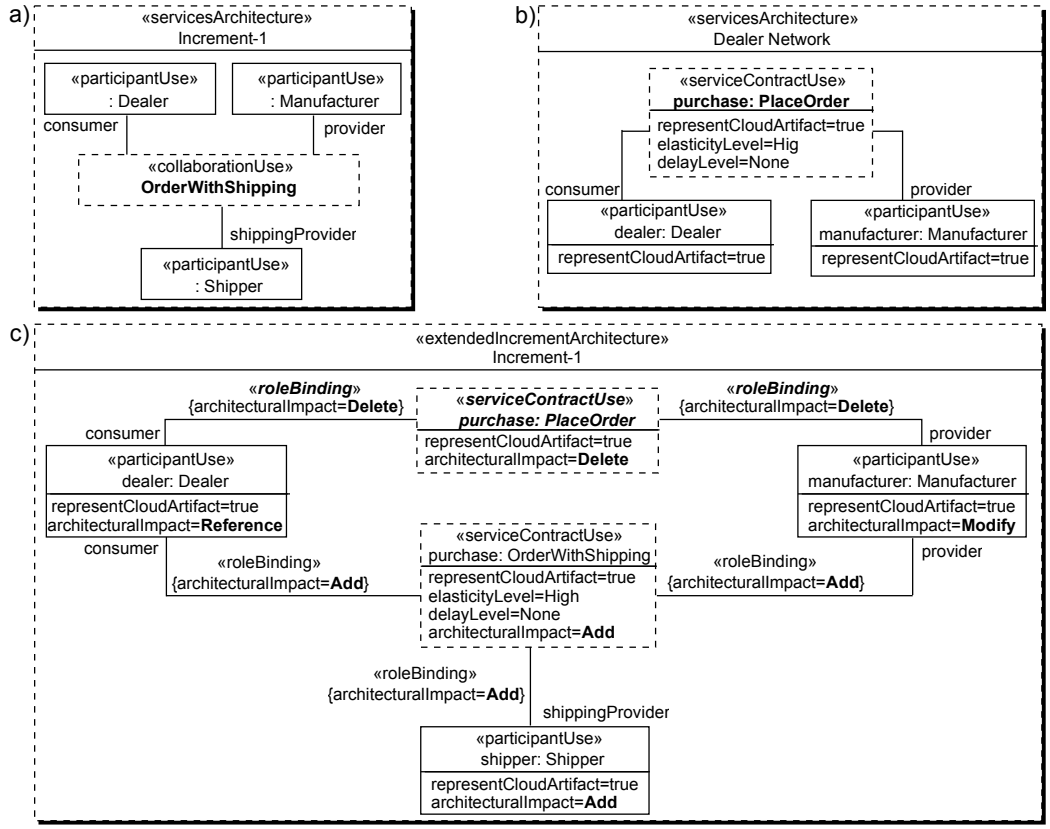

Fig. 4. (a) Increment Architecture Model, (b) Current Architecture Model, (c) Extended Increment Architecture Model

The output of this activity was the *Extended Increment Architecture Model*. An explanation of how this activity output was used to generate the cloud artifacts that facilitate the dynamic architecture reconfiguration is provided in the following section.

6.2. Supporting the *Reconfigure Architecture* Activity

In this activity, we implemented the *Cloud Artifacts Model* as an Ecore model in EMF, and defined model transformations using ATLAS Transformation Language (ATL) to generate it (Fig. 5a). The model transformation alternative that was applied was selected according to Table 1, taking into account that the «*ServiceContractUse*» purchase:OrderWithShipping (see Fig. 4c) requires *elasticityLevel=High* and *delayLevel=None.*

After generating the *Cloud Artifacts Model*, it was completed by using the EMF Ecore editor. *Setting* classes were created to specify deployment configurations (e.g., ports that must be opened, certificates that must be installed), along with service configurations (e.g., the Role Instances setting's initial value was set at 2 in order to satisfy the *elasticityLevel=High* requirement). Once completed, the M2T transformations generated cloud artifacts that were specific to a cloud platform; Table 2 shows the mapping between the *Cloud Artifacts Model*'s metamodel classes and the cloud artifacts required to implement their instances in Microsoft Azure. We chose to deploy cloud artifacts in Microsoft Azure because it is better suited to service-oriented-architecture based applications [19]. The cloud artifacts generated were therefore .NET/WFC-compliant (Windows Communication Foundation), while Visual Studio 2013 was used as the development environment.

Table 2. Excerpt of the mapping among architectural elements, Cloud Artifacts Model's metamodel classes and cloud artifacts used in Microsoft Azure.

Architectural element	Cloud Artifacts Model's metamodel classes	Cloud artifacts in Microsoft Azure
Service Contract	Deployment Project	Azure Cloud Service Project
	DynamicConfiguration	ServiceConfiguration.Cloud.cscfg
	Interaction Project	WCF Service Web Role Project
	MessageType	Class decorated with [Message Contract]
	DataType	DataContract/Class/Enumeration
	Interface	Interface decorated with [ServiceContract]
	OrchestrationService	WCF Workflow Service
	DynamicConfiguration	Web.config

Web Service Description Language (WSDL) files are used to describe services and allow interoperability between platforms. We chose to define M2T transformations to generate WCF-compliant artifacts (classes) and employed Visual Studio to generate WSDL files from these classes rather than define M2T transformations to generate WSDL files; however, our metamodel supports both approaches by using the *representation* attribute from the *Artifact* class. We did not generate WSDL files because when imported into a development environment, it generates its own WSDL representation which differs from the original, signifying that developers still need to do additional work in order to make the tool to generate a WSDL that is able to interact with other platforms [18]. We similarly generated WCF Workflows Services to implement orchestrations rather than using the Business Process Execution Language (BPEL).

Fig. 5b shows the cloud artifacts that implement the architectural element «*ServiceContract*» OrderWithShipping as an orchestration service, in accordance with the cloud artifacts organization described in the previously generated *Cloud Artifacts Model* (see Fig. 5a).

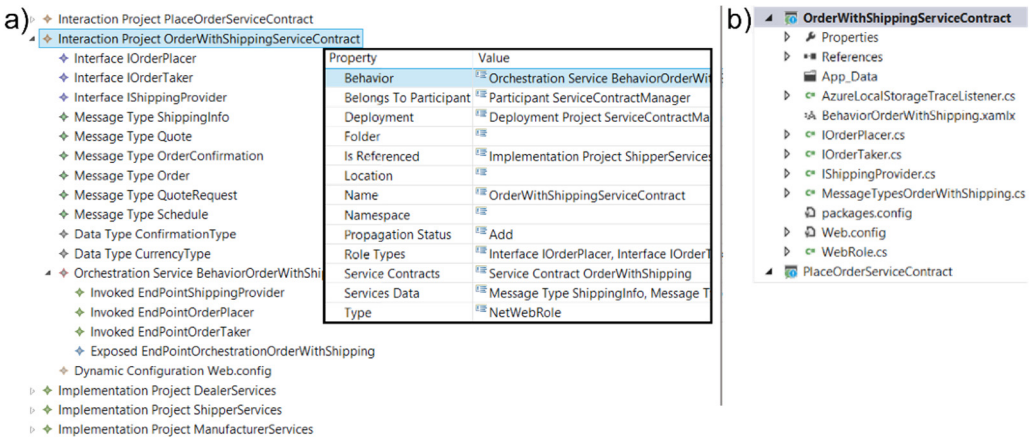

Fig. 5. (a) Screenshot of an excerpt of the Clod Artifacts Model corresponding to Increment-1, (b) Screenshot of the cloud artifacts that implement the OrderWithShipping service

After deploying the artifacts generated, *Invoked EndPoints* in the *Cloud Artifacts Model* related to the orchestration cloud service (OrchestrationServiceBehaviorOrderWithShipping) were updated with information corresponding to the *Exposed EndPoints* of cloud services that participate in the interaction. In order to allow the communication with the new orchestration service, the *Invoked EndPoint* of the cloud service belonging to the participant that initiates interaction (Dealer's *Front End Service*) was simultaneously updated with information from the *Exposed EndPoint* corresponding to the orchestration cloud service. Once updated, the M2T transformations generated the cloud artifacts that make up the *Reconfiguration Plan Specific for Cloud Provider,* which included some XML Document Transform (XDT) files used in Visual Studio to modify service configuration files while the deployment takes place. The following source code is an excerpt of the XDT file generated to update the Dealer's *Front End Service* configuration, changing its binding information (*EndPointPurchase*) at runtime:

```
<ServiceConfiguration serviceName="Dealer"
      xmlns:xdt="http://schemas.microsoft.com/
      XML-Document-Transform">
      <Role name="DealerServices">
            <ConfigurationSettings>
                  <Setting name="EndPointPurchase"
                        value="http://../OrderWithShipping.xamlx"
                        xdt:Transform="Replace"
                        xdt:Locator="Match(name)" />
            </ConfigurationSettings>
      </Role>
</ServiceConfiguration>
```

Finally, the M2M transformations updated the *Current Architectural Model* (see Fig. 6).

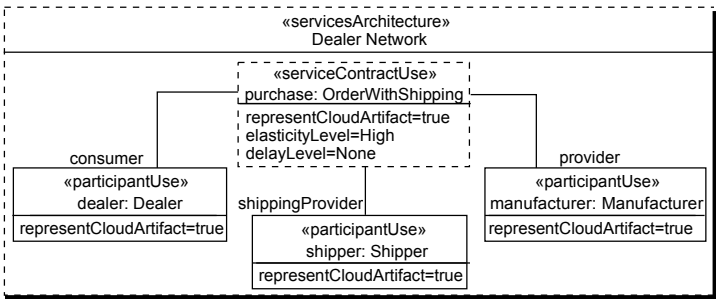

Fig. 6. Current Architecture Model after integration of Increment-1

7. Conclusions and Further Work

In this paper, we have presented an MDD approach with which to support the dynamic reconfiguration of cloud application architectures caused by the integration of new cloud services. The architectural reconfiguration is achieved by replacing the orchestration of services and bindings to the orchestrated services at runtime.

In order to specify how the integration of new services will change the current cloud application architecture, we extended SoaML and defined the *DIARy-specification-profile*. We believe that this profile will improve the current increment specification practices since it allows software architects to: i) avoid working with the overall cloud application architecture model when specifying the increment integration; and ii) plan the reconfiguration actions that will occur when deploying a cloud service increment beforehand. We have also proposed the *Cloud Artifacts Model,* which is a high-level description of how cloud artifacts should be organized and which cloud platform services should be used in order to implement increment integration specifications. This model promotes: i) the application of service-oriented principles during implementation, organizing cloud artifacts into projects that can be deployed on various platforms; ii) the decoupling of software artifacts that implement orchestration from those that implement participants' internal process, thus signifying that the orchestration of services can be offered as a service and easing the architectural reconfiguration.

We have shown the feasibility of our proposal by applying it to a case study. We are currently working on a prototype that can be used to implement alternative transformation chains and reconfiguration services required at runtime. As further work, we plan to abstract adaptation patterns in order to represent prescriptions of the steps required to operationalize reconfiguration actions at a cloud platform-independent level, and to provide tool support in order to integrate the technologies used. We also plan to design experiments with which to validate the effectiveness of our approach in practice.

Acknowledgements

This research is supported by the Value@Cloud project (MINECO TIN2013-46300-R); Facultad de Ingeniería, Universidad de Cuenca-Ecuador; and the Microsoft Azure Research Awards.

References

[1] Ardagna, D., Nitto, E. Di, Milano, P., Petcu, D., Sheridan, C., Ballagny, C., Andria, F. D., and Matthews, P.2012, "MODACLOUDS : A Model-Driven Approach for the Design and Execution of Applications on Multiple Clouds," pp. 50–56.

[2] Bergmayr, A., Troya, J., Neubauer, P., Wimmer, M., and Kappel, G.2014, "UML-based Cloud Application Modeling with Libraries , Profiles , and Templates," in In Proc. Workshop on CloudMDE, pp. 56–65.

[3] Berre, A. J.2008, "Service Oriented Architecture Modeling Language (SoaML)-Specification for the UML Profile and Metamodel for Services," Object Manag. Group.

[4] Brandtzæg, E., Mosser, S., and Mohagheghi, P.2012, "Towards CloudML, a Model-based Approach to Provision Resources in the Clouds," in ECMFA, pp. 18–27.

[5] Breivold, H. P., Crnkovic, I., Radosevic, I., and Balatinac, I.2014, "Architecting for the Cloud: A Systematic Review," Int. Conf. on Computational Science and Engineering.

[6] Brogi, A., Ibrahim, A., Soldani, J., Carrasco, J., Cubo, J., Pimentel, E., and D'Andria, F.Feb. 2014, "SeaClouds," ACM SIGSOFT Softw. Eng. Notes, vol. 39, no. 1, pp. 1–4.

[7] Cámara, J., Martin, J. A., Salaun, G., Cubo, J., Ouederni, M., Canal, C., and Pimentel, E.2009, "Itaca: An Integrated Toolbox for the Automatic Composition and Adaptation of Web Services," in 31st Int. Conf. on Software Engineering, pp. 627 – 630.

[8] Ermagan, V. and Krüger, I.2007, "A UML2 Profile for Service Modeling," Model Driven Eng. Lang. Syst., vol. 4735, pp. 360–374.

[9] Fehling, C., Leymann, F., Retter, R., Schupeck, W., and Arbitter, P.Cloud Computing Patterns: Fundamentals to Design, Build and Manage Cloud Apps. 2014, Springer.

[10] Gomaa, H., Hashimoto, K., Kim, M., Malek, S., and Menascé, D.2010, "Software Adaptation Patterns for Service-Oriented Architectures," in ACM Symposium on Applied Computing, pp. 462–469.

[11] Guillén, J., Miranda, J., Murillo, J. M., and Canal, C.2013, "A UML Profile for Modeling Multicloud Applicat.," in Service-Oriented and Cloud Computing.

[12] Homer, A., Sharp, J., Brader, L., Narumoto, M., and Swanson, T.Cloud Design Patterns: Prescriptive Architecture Guidance. 2014, Microsoft patterns & Practices.

[13] IEEE.2000, "Recommended Practice for Architectural Description of Software-Intensive Systems," (IEEE Std 1471-2000), IEEE Computer Society.

[14] Insfrán, E., Gonzalez-Huerta, J., and Abrahão, S.2010, "Design Guidelines for the Develop. of Quality-Driven Model Transformations," MoDELS, pp. 288–302.

[15] Johnston, S.2005, "UML 2.0 Profile for Software Services," IBM Dev. http//www. ibm. com/developerworks/rational/library/05/419_soa.

[16] Lane, S., Gu, Q., Lago, P., and Richardson, I.2013, "A Pragmatic Approach for Anal. and Design of Service Inventories," in Service Oriented Computing and Applicat.

[17] Malavolta, I., Lago, P., Muccini, H., Pelliccione, P., and Tang, A.2013, "What Ind. Needs from Architectural Languages," IEEE Trans. Softw. Eng., vol. 39, no. 6.

[18] Mcgregor, S., Russ, T., Wilde, N., Gabes, J. P., Hutchinson, W., Duhon, D., and Raza, A.2012, "Experiences Implementing Interoperable SOA in a Security-Conscious Env."

[19] Muppalla, A. K., Pramod, N., and Srinivasa, K.2013, "Efficient Practices and Frameworks for Cloud-Based Application Development," in Software Engineering Frameworks for the Cloud Computing Paradigm, Springer, pp. 305–329.

[20] Parameswaran, a. V. and Chaddha, a.2009, "Cloud Interoperability and Standardization," Infosys Technol. Limited/SETLabs Briefings, vol. 7, no. 7.

[21] Perepletchikov, M., Ryan, C., Frampton, K., and Schmidt, H.2008, "Formalising Service-Oriented Design," J. Softw., vol. 3, no. 2, pp. 1–14.

[22] Todoran, I., Hussain, Z., and Gromov, N.Aug. 2011, "SOA Integration Modeling: An Evaluation of how SoaML Completes UML Modeling," IEEE 15th Int .Enterprise Distrib. Object Comput. Conf. Work., pp. 57–66.

[23] Zuñiga-Prieto, M., Gonzalez-Huerta, J., Abrahao, S., and Insfran, E.2014, "Towards a Model-Driven Dynamic Architecture Reconfiguration Process for Cloud Services Integration," in 8th Int. Workshop on Models and Evolution co-located with MODELS.