Supporting Elastic Cloud Computation with Intention Description Language

T. Baker, A. Hussien, M. Randles, A. Taleb-Bendiab,
School of Computing and Mathematical Sciences
Liverpool John Moores University
Byrom Street, Liverpool, L3 3AF, UK
t.baker@ljm.ac.uk, a.hussien@ljmu.ac.uk, m.j.randles@ljmu.ac.uk, a.talebend@ljm.ac.uk

Abstract- The continuing shift towards cloud computing is a major influence in today’s industry. The proliferation and availability of cloud computing facilities, including on-demand platforms, on-premises platforms and Platform as a Service (SaaS), is a key part of that shift. Theoretically, therefore, it ought to be straightforward to create a new elastic Cloud-Based Application (CBA) because of the tools and techniques within the available platforms. Moreover, at design time, an application can be enhanced by accessing/composing services provided by the cloud rather than developing entirely new services. Practically and programmatically speaking, however, there is still huge gap between the users and the cloud, as the clouds’ users are still not able to dynamically update their applications’ components and services at runtime in an effective manner. Attaching a service into a running CBA to enhance the application behaviour requires some knowledge of at least the service and the application source code so that code composition can be take place. This paper demonstrates a new Intention Description Language (IDL) that can be used via Provision, Assurance, and Accounting (PAA) modelling approach, to weave a new service to the elastic CBA at runtime. The IDL aims at bridging (how, why and what) is done by representing the business processes managed by business rules. While the PAA description is out of the scope of this paper, a full explanation of IDL is presented here.

I. INTRODUCTION

The continuing move to cloud computing platforms represents a major upheaval in the computer industry. These kinds of platforms are becoming widely used in dynamic economic environments, where a company’s survival depends on its applications’ status and the applications admin’s ability to focus on core business needs and adapt the application quickly at work time by adding, removing and modifying any of the web services or components based on the BPEL and WSDL [1, 2].

However, the Web Service Description Language (WSDL) [3, 7] was developed to describe the technical details, (partners, partner link types and port types), to show how a web service can be accessed and invoked remotely over the web to enhance or expand systems functionalities. In the same regard, state of the art business process automation systems are based on Business Process Execution Language (BPEL) [4] as services composition language, which relies on WSDL for describing and automating business behaviour. Since the WSDL was made to be used by computers, it describes only how something can be done technically [5], e.g. how a web service can be invoked.

From the elasticity and scalability perspectives, the dynamic Cloud-Based Application (CBA) has different requirements from the ones fulfilled with WSDL and BPEL. While the technical description of services is important in the cloud computing [6, 7], describing why and what is done are also important.

In this paper, we proposed a new Intention Description Language (IDL) that can be used via Provision, Assurance, and Accounting (PAA) modelling approach, to weave the CBA at runtime. The IDL therefore aims at bridging how, why and what is done by representing the business processes managed by business rules. While the PAA description is out of the scope of this paper and can be found in [8, 9, 10], a full explanation of IDL is presented here.

The remainder of this paper is structured in five main sections. Section 2 gives more details about the problem statement this paper intended to solve. The proposed solution is given in section 3 with a precise description about the IDL meta-model. A case study, PAAPetShop, is available in section 4 to aid a clearer understanding of the proposed approach from a practical point of view. However, to, evaluate this approach at runtime, it is necessary to demonstrate its practicality and reliability, this will be explained in section 5. The paper ends with some conclusions and future work.

II. PROBLEM STATEMENT

The main characteristic of Elastic Compute Cloud (EC2), which has promoted its use amongst many of the largest businesses for most, if not all, of their business applications, is the elasticity of this computing paradigm [11]. Elasticity means the CBAs, especially long running applications, should be flexible and adaptable to accept new changes and requirements from different parties at runtime in an
automated manner. The requirements might need to invoke and inject services and components from other platforms using different description languages (e.g. WSDL, BPEL…etc) to achieve the desired behaviour.

WSDL is designed to provide the web services technical details that are used and needed by BPEL to invoke and execute the web services. While the technical details alone are not enough to achieve the full cloud elasticity, the BPEL itself is too rigid to cope with changing business demands instantly [5], without the need for taking the system offline to modify the business demands.

However, the elastic cloud computing applications need formal/standard description language support to allow these applications to be dynamically updated by integrating services and components from other languages and frameworks at runtime.

III. PROPOSED SOLUTION

To overcome the above problem, we propose a more pragmatic weaving approach that is mainly composed of two parts: A CBAs IDL, that sits in the cloud and is accessible and easily adapted, by cloud users, at runtime, and the PAA framework (SaaS) that also sits in the cloud that ought to efficiently read and execute IDL and the emergent requirements expressed through the using of IDL. While the PAA framework has been widely explained in [8, 9, 10], this paper focuses on detailing the IDL.

A. Intention Description Language (IDL)

Tangible products usually have a well-defined set of possible variants for customization. For example, if a customer requires a faster PC, more powerful CPU and increased RAM can be designed, built and attached to the computer motherboard.

However, the same cannot be straightforwardly achieved for intangible cloud applications/services/components. This makes the need for a comprehensive description language one of the most important undertakings for the elastic cloud computing. Thus, the cloud elasticity requires combining and correlating business, operational and IT aspects into CBA IDL.

B. IDL Approaches

IDL is an xml CBA description language. It uses three main approaches to model and automate business behaviour on cloud applications.

The first one focuses on flows of activities that generate values. This part of IDL is called the flow model. The flow model is used to describe “what” is happening; representing this in the xml using the Flow attribute, which is encapsulated within each task to direct the execution to the next task. The flow, as shown in Listing 1 has two main subclasses move to and the decision. Where <move> directs the IDL to the next task, whereas <Decision> is used to direct the IDL to a sudden task that arises according to the user’s emergent requirements. The two types are shown in IDL Listing 1.

Listing 1: IDL flow model

```
<Listing 1: IDL flow model>
```

The second approach uses rules/requirements to describe desired business behaviour. This so called Business Rules/Requirements (BR) approach uses rules/requirements to describe “why” something has to happen according to the rules and the requirements, and uses technologies to automate decision logic. The NeptuneScript is used in the IDL to describe and create executable functions that describe why something is done according to the requirements.

The following, Listing 2, highlights this part.

Listing 2: IDL BR

```
<Listing 2: IDL BR>
```
The third approach is the technical specification of the services and components that should be used to accomplish certain behaviour. The technical specification of the service is, for example, the code it uses; the name of the service, the type of the service, the URL address of the service, the execution engine...etc. which is shown in the following, Listing 3.

Listing 3: IDL technical specification

```
<complete id="2" code="# WSDL" execution_engine="# BPEL">
  <nbo name="_default" type="executable" service="# sellPets" />
  <nbo name="_default" type="executable" service="# buyPets" />
</complete>
```

In this case, the IDL can be described as a comprehensive description language due to its nature of integrating the three different approaches (what, how and why).

Thus, the IDL schema defines three core types of information that provide the descriptions that a consumer can use to discover, select, invoke services and have a view on services’ behaviour at execution time.

**C. Formalizing IDL Meta-model**

In order to establish a proper base for IDL to be used in the design and creation of CBAs, we provide a formal specification for it. This serves for purposes such as communication and implementation for integration with other specifications and languages like WSDL and BPEL.

As explained in the previous sections, the IDL consists mainly of a set of processes that describe the business behaviours. Some of these processes are composed of sub processes which both include a task or set of tasks that should be executed to achieve the desired behaviour of that process. The process model information is attached to the IDL to provide full information about the current IDL, for example, the IDL creator, IDL name, IDL owner, and the guide key.

As shown in Figure 1, there are different kinds of requirements that are defined in the IDL to give a clear view to the user and to satisfy the three different approaches explained above. The Provision requirements show the specific user requirements and the individual services and components the user wants to use in the application at runtime. However, the provision requirements can either be added manually by the user or extracted automatically by the provision generator at runtime during the application execution.

The Assurance requirements describe properties that are fundamental for the characterization of a service. We rely on a set of non-functional properties such as validity, availability, service type...etc. In order to provide a suitable language that can be understood by business stakeholders and consumers, this requirement has been extended to include Concept Aided-Situation Prediction Action (CA-SPA) policy, which can be used to add new prediction and action at runtime according to the current situation; more information about CA-SPA can be found in [12].

The Accounting requirements represent the statistical side of the application and supply the dashboard with the necessary information to allow better managing of the application at runtime.

The following in Listing 4 shows the IDL Provision Assurance and Accounting requirements which have been added at runtime to the PetShop IDL, described in the next section, to add a new process (check delivery) and validate the address of the shipped order.
Listing 4: PAA requirements
-------------------------------------------------------------------
define ValidPreShippingOrderPAA as PAA
{
    provisioning
    {
        order as PetOrderDetails
    }
    assurance
    {
        order.ShippingAddress is VALID
        order.SelectedPet is INSTOCK
        order.ShippingDate is NULL
    }
    Accounting
    {
        /* error handling or logging defined here*/
    }
}-------------------------------------------------------------------

IV. CASE STUDY: PETSHOP IDL

PetShop is an architectural blueprint developed by Microsoft based on the original Sun Microsystems PetStore benchmark enterprise architecture for e-commerce and enterprise systems [13]. The application produced by the PetShop blueprint builds a web-based application for browsing and purchasing pets.

In particular, the PetShop application is implemented from Web Services composition that perform the main processes outlined in Figure 2, which for instance enable administrators of the system to add, remove, and modify the animals available for sale, with their data stored in a database.

To this end, this section outlines the adaptation processes when a new process or service can be injected (or simply added) at runtime to the standard PetShop process model.

The standard PetShop blueprint does not support EC2, as it is a static design approach. In other words, the process model cannot be adapted at runtime (e.g. add a new service to the PetShop process model like PetRefund). The whole system would need to be taken offline before the adaptation, and then re-uploading it again online after the adaptation. Of course, the system administrator should undertake this. So the user will not be able to add any new function or service to the original process model on the cloud.

Hence, there is a new way for designing the PetShop via PAA, using the PetShop cloud IDL which can be accessed and modified by the user via the PAA editor service; which allows the user to modify the entire process model and upload the modified version at runtime. The PAA PetShop user interface is shown in Figure 3.

Figure 3: PAA PetShop-based-Cloud

The user will be mainly dealing with the above interface, s/he can move from this page to another PetShop page using the available buttons. The movement from this page to the next one is managed and controlled by IDL flow model. The IDL might be saved on another virtual machine on the cloud, and can be accessed and used via the intermediate model, shown in Listing 5.

Thus, updating Figure 2, the PetShop process model, can be completed using the Edit button, which displays another web page for the PAA Wiki Editor for IDL, as shown in the Figure 4.

Figure 4: IDL runtime editor

More information about the PAA wiki Editor and how to update the process model can be found in [14].
A. PetShop CBA and IDL interconnection

The PAA PetShop itself can be saved on another Virtual Machine (VM) on the cloud. All the information needed by PAA PetShop to access and use the IDL is saved in another intermediate model, which is part of the PAA PetShop. This intermediate model saves all the IDL information that is shown in the IDL meta model (the IDL creator, IDL name, IDL owner, and the guide key). This information is used to differentiate the requested IDL from the other IDL models on the cloud.

Figure 5: High-level overview of the CBAs IDL

Listing 5: IDL intermediate model

```xml
<model version="1.0" encoding="utf-8">
<IDLModelStub>
  <Name>PetShop</Name>
  <ID>26a736c-62c8-4eb4-b82e-7f11d29050c8</ID>
  <IDLModelFileName>c:\inetpub\wwwroot\neptune\DecisionModel\PetShop.xml</IDLModelFileName>
  <Organisation>LJMU</Organisation>
  <Creator>Thar Baker</Creator>
  <DateAdded>2009-05-07T13:56:17.8118415+00:00</DateAdded>
  <DateAccessed>2009-05-07T13:56:17.8118415+00:00</DateAccessed>
</IDLModelStub>

<IDLModelStub>
  <Name>Simple Triage</Name>
  <ID>F1482f1e-1a31-4a5f-b3c5-0b3a7b4f6df3</ID>
  <IDLModelFileName>c:\inetpub\wwwroot\neptune\DecisionModel\swellingsimple.xml</IDLModelFileName>
  <Organisation>LJMU</Organisation>
  <DateAdded>2009-05-07T09:44:45.9107592+00:00</DateAdded>
  <DateAccessed>2009-05-07T09:44:45.9107592+00:00</DateAccessed>
</IDLModelStub>

<IDLModelStub>
  <Name>Simple Triage Extended</Name>
  <ID>4537d171-1e1e-4b09-9c7a-d48bae851282</ID>
  <IDLModelFileName>c:\inetpub\wwwroot\neptune\DecisionModel\extended.xml</IDLModelFileName>
  <DateAdded>2009-05-07T09:51:06.4708236+00:00</DateAdded>
  <DateAccessed>2009-05-07T09:51:06.4708236+00:00</DateAccessed>
</IDLModelStub>

</model>
```

Hence, the interconnection between the PAA PetShop and the IDL is established; whether they are both saved on the same cloud, or another public/private cloud, can be determined through the intermediate model. The following Figure 5 shows a high level overview of the proposed elastic CBAs via the use of IDL.

As shown in Listing 5, the intermediate model is an xml-based model. The agility behind this model is it provides important data in the form of adaptable xml tags. These tags can be adapted at runtime, by the user, using PAA Wiki Editor [8, 14]. Hence, accordingly, the IDL model and the other information can be adapted at runtime depending on the location of the requested IDL. On the other hand, this can provide insights to the user or the system admin to track user history by considering the date that the IDL has been used and the time of the last access into it.

V. Evaluation: System Scalability

A test was conducted to ascertain the scalability of CBA PAA PetShop during its operation in comparison to the original Microsoft PetShop. By introducing a new behaviour to PAA PetShop via IDL, the performance impact of the further interpretation of the new behaviour could be contrasted against that of the performance impact of the introduction of the same behaviour in the Microsoft PetShop model.

Figure 6 shows two systems executed to produce the same behaviours over a time-scale of cycles of a process execution. After 50,000 executions, a new task requirement is introduced in the IDL model, and a task description updated accordingly. As new semantic linking needs to take place, performance is reduced, such that the time to complete the process increases. It should be noted, however, that as only one new requirement is needed to be linked, the performance impact is less than that of the original initiation at cycle 0, where many new requirements are introduced at the same time. After the reconfiguration, linking and execution, performance returns to a new standard, slightly slower than the original behaviour from 10,000 to 50,000. This is due to the added time needed to execute the action by Neptune.
On the other hand, there is a slight increase in execution time of the original PetShop (given in green) from 50,000, due to the new behaviour introduced to the PetShop code to be executed. It can be noted that after the execution of the new behaviour, however, the execution performance of PAA PetShop compared to Microsoft PetShop is largely similar, thus scalability issues can be said to be introduced.

Figure 6: Scalability testing of PAA PetShop vs. Microsoft PetShop

VI. CONCLUSIONS AND FUTURE WORKS

This paper has argued for the design and use of an adaptable intention description language for cloud-based applications to achieve the cloud elasticity characteristics through the using of the PAA approach. This means it will be straightforward, quick and reliable to amend the cloud-based applications at run time without the need for the recoding and republishing of the application.

The perceived value of using IDL via PAA Neptune has driven the re-development of PetShop to be a Cloud-Based Application through this approach.

In future work, the IDL description language is to be rigorously tested and evaluated to ensure the design of IDL and PAA tools is correctly envisaged. But there is still a large body of outstanding research issues, which have to be resolved before the full benefits of exploitation can be realised.

REFERENCES


Nov. 25, 2009).


