Computer Standards & Interfaces xxx (2015) xxx–xxx

Contents lists available at ScienceDirect

Computer Standards & Interfaces

journal homepage: www.elsevier.com/locate/csi

A cloud service for COTS component-based architectures

Jesús Vallecillos, Javier Criado, Nicolás Padilla, Luis Iribarne

Applied Computing Group, University of Almería, Spain

ARTICLE INFO

Article history:
Received 3 February 2015
Received in revised form 16 November 2015
Accepted 28 November 2015
Available online xxxx

Keywords:
Cloud service
Software architectures
Component-based systems
COTS

ABSTRACT

Software architecture management, especially in component-based web user interfaces is important to enhance their run-time accessibility, dynamics and management. The cloud offers some excellent mechanisms for this kind of systems, since software can be managed remotely, easy availability of the resources is ensured and mass storage is possible. This article presents an infrastructure solution, based on the use of web services and cloud computing, for managing COTS-based architectures.

1. Introduction

In general, software available on the web must increasingly be changed, updated and adapted to user demands. Such software is sometimes built up from components, or component-based architectures are used to describe its structure. In both approaches, they must be accessible at any time, dynamic, managed at run-time and adaptable to changes [5]. Web services and cloud computing offer an excellent infrastructure for this, and since the software can be managed remotely, high resource availability is ensured and mass storage is possible. An example of such architectures is component-based web interfaces, which have the same requirements, and must be dynamic and adapt thier structure to the user preferences. With this aim, new projects and proposals have come up in the last few years to build customized web *User Interfaces* (UI) by configuring the widgets the user wants to visualize [25]. For these applications, the user has a *Graphical User Interface* (GUI) available that can be configured to create a dashboard. This type of GUI is built from graphical components of high or medium granularity (that is, they are not simple buttons or text fields) that group together some functionalities related to each other and make up widget-based mashup applications [46,15], such as MyYahoo, Ducksboard or Netvibes [42].

This led to our interest in developing an infrastructure for managing component-based software architectures. In particular, our research work is focused on dynamic management of component-based UIs. The three pillars on which our proposal is based are therefore: CBSE (*Component-based Software Engineering*), MDE (*Model-Driven Engineering*) and *Cloud Computing*. **Component-based Software Engineering** [11] is a software engineering discipline that improves software development by reusing it, contributing reliability, and reducing the time required for creating such software. Contrary to traditional software development, CBSE is focused on integrating previously developed software components into the system following a bottom-up development instead of a traditional top-down perspective. The concept of component reuse and management is also present in standards [2]. Our proposal requires that the user interface be defined as a set of components, in which each application component represents an individual user interface component. This proposal follows a bottom-up perspective for developing (at run-time) the UI structure from components available in one or more third-party repositories.

The UI components used in our proposal are called COTSgets, from COTS (*Commercial Off-The-Shelf*) [28] and gadgets (understood as any software that can work alone or as a piece of the architecture). A COTS component is any coarse-grained component developed by third parties available for building more complex systems. An example of a GUI application developed by us under a Project of Excellence funded by the Junta de Andalucía [Andalusian Government] [18] may be seen at http://acg.ual.es/enia/COTSbasedArchitectureExample. This real application will show the reader what a COTSget-based architecture looks like. It can be tried out, and hopefully, the role of this kind of component will be easier to understand.

The second pillar is **Model-Driven Engineering**. This engineering discipline is focused on constructing models on different levels of abstraction, facilitating software specification, and providing several mechanisms for automating the development of the final product using of model transformation techniques. Some systems developed with these techniques attempt to provide software with adaptive capabilities for dynamic reconfiguration of the models at run-time, so they may act differently as the adapt to the circumstances of their execution, such as changes in the user interaction, available resources, or execution platforms [37,16]. In the particular domain of component-based software systems, MDE techniques can facilitate architecture design and

http://dx.doi.org/10.1016/j.csi.2015.11.008
0920-5489/© 2015 Elsevier B.V. All rights reserved.

development, for example, in defining their structure, component behavior and relationships, interaction, or their functional and non-functional properties [12]. Furthermore, adaptation of architectural models at run-time makes it possible to generate different software systems based on the same abstract definition, for example, to cope with different user preferences, component status or target platforms [3]. Fig. 1 shows how our component-based architectures can be represented on three levels of abstraction (in the infrastructure proposed):

- Abstract architectural model, corresponding to the Platform Independent Model (PIM) level in Model Driven Architecture (MDA) [31], and representing the architecture in terms of the type of components it contains and their relationships.
- Concrete architectural model, corresponding to the Platform Specific Model (PSM) level, and describing the concrete components that comply with the definition of the abstract architecture.
- Final software architecture, which represents the source code (our components) to be executed or interpreted.

Run-time adaptation of the architectures is based on processes executed on the abstract and concrete architecture levels [10,27]. On the abstract level, model-to-model transformation processes [14] are executed to adapt the abstract architectural models to changes in context [9]. On the concrete level, the concrete architectural models are realized by a trading process [28,8], which calculates the configurations of concrete components that best meet the abstract definitions. This provides the possibility of generating different software architectures based on the same abstract definition, for example so it can be executed on different platforms. The content of this paper focuses only on showing the technological infrastructure used on the concrete level and the final architecture. Adaptation on the abstract level (PIM perspective), the *trading* process for the concrete architectures (PSM), synchronization of abstract models and final architectures, or how the changes in the models affect the executing architecture are outside the scope of this paper.

The third pillar is **Cloud Computing**. The strengths of cloud computing for users and organizations have been widely described in the literature, *e.g.*, [33] or [44]. The benefits identified include the use of *Software-as-a-Service* (SaaS) and specifically *Models-as-a-Service* (MaaS) as on-demand high-level abstract software. The combined use of MaaS and MDE in turn has many benefits [6] to highlight such aspects as their availability, run-time sharing, improved scalability and distribution, *etc*. In our proposal, instead of proposing general use of this concept, our work focuses on the management of software architectures based on our COTSget components. Therefore, inspired by the use of these components in models as services and as a mechanism for access to these models through web services deployed in the cloud, we have created a cloud service called *COTSgets-as-a-Service*. To provide this service, a cloud infrastructure organized in three layers has been created (Fig. 1): the client layer (C), the platform-dependent server layer (B) and the platform-independent server layer (A).

The client layer is made up of user applications, and therefore, it comprises the set of components defining the final software architecture. Currently, all the developed applications are deployed on the web platform and have been implemented using widgets, following the recommendation of W3C. The platform-dependent layer is intended to (1) provide the client with the required services and (2) interact with the independent layer, thus obtaining some services from it and providing it with others. The platform-independent layer offers those services which are valid for all platforms. These services are based only on the description of components and their relationships, regardless of the platform where components will be deployed.

This article is based on previous research work [43] in the field of distributed development of information systems [34]. In that work, the system infrastructure focusing on the three-level data model used in the different layers of our architecture is described. The work presented in this manuscript focuses on the technological infrastructure based on web services and cloud computing used for the deployment of COTS component-based architectures.

The rest of the paper is organized as follows: Section 2 shows an example of an scenario where the most important concepts used in the rest of the article are explained. Section 3 describes the languages proposed (using an MDE perspective) for describing COTSget-based architectures. Section 4 explains how the *COTSgets-as-a-Service* is provided using a cloud service. Section 5 illustrates the process used to validate and evaluate the cloud service and its performance. Section 6 presents the related work and, finally, Section 7 summarizes the conclusions and discusses the future work.

2. An example scenario

This section describes a web-based application as a sample scenario to explain many of the concepts used in the rest of the article. This application (Fig. 2) was dynamically constructed from COTSgets components for the abovementioned ENIA research project. The application deals with a geographic information query system, with which visual layers with this type of information can be loaded. These layers provide data acquired from a set of *Open Geospatial Consortium* (OGC) services provided by the REDIAM (*Environmental Information Network of Andalusia*).

The components in this application are not assembled alone (independently of each other), but rather coupled, as described below, to help designers build complex interactive applications. In the upper right-hand corner of the figure, there are two components: *UserInfo* and *Logout*. The first one is responsible for identifying the user who has connected and showing user-specific information, such as the profile they are currently using in the system, and the *Logout* component closes that user's session. These two components are not isolated from each other. In fact, there is a component called *Header*, which in

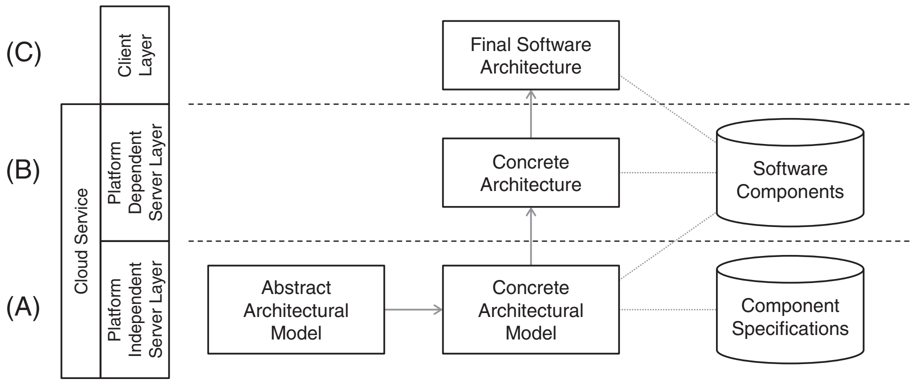
(C)
Client Layer
(B)
Platform Dependent Server Layer
(A)
Platform Independent Server Layer
Cloud Service
Final Software Architecture
Concrete Architecture
Software Components
Abstract Architectural Model
Concrete Architectural Model
Component Specifications

Fig. 1. Abstraction levels and layers of the proposed infrastructure.