A new cloud computing architecture for music composition

Jesús L. Alvaro, Beatriz Barros*

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga, Campus de Teatinos, Bulevar Louis Pasteur, 35, 29071 Málaga, Spain

A R T I C L E  I N F O
Article history:
Received 18 October 2011
Received in revised form 28 March 2012
Accepted 27 April 2012
Available online 11 May 2012
Keywords:
Cloud computing
Music composition
Music representation
Algorithmic composition
Computer music

A B S T R A C T
This paper presents an original cloud computing architecture for music composition. In this model, music applications are built by making several computer music services work together. Component services are provided by a dedicated layer in the cloud architecture called computer music as a Service (CMaaS). The specialized music services can be integrated into different applications at the same time. These music services provided by the CMaaS layer are implemented in the form of platform images based on templates at the Platform as a Service layer. The images are ready to be loaded into the virtualized infrastructure on demand. As examples of implementation over the proposed cloud architecture, two powerful applications for computer music composition are presented: “Diatonic Composer”, an interactive composer of scores with high-abstraction music elements, and “Csound Meets the Cloud”, an assisted algorithmic composer focused on sound synthesis. The composition model, the involved music services and the web application are described for the above mentioned applications. The proposed architecture, the implemented services and the provided application examples constitute a decided step towards distributed music computation.

© 2012 Elsevier Ltd. All rights reserved.

1. Introduction

Cloud computing has turned internet into a large repository of specialized services (Lombardi and DiPietro, 2011). Some Hypermedia applications working on clouds related to graphics (Shi et al., 2011) and video multicast (Shen et al., 2011) have been reported. This paper presents a new cloud computing architecture for computer music focused on music composition.

Computer Music (CM) is an inter-disciplinary domain where Psychology, Acoustics, Engineering, Computing and Music meet (Moore, 1990). Within CM, Computer music composition focuses on those aspects related to music composition and the use of Information Technologies (IT). Computers have been offering composers valuable assistance, from score notation to sound synthesis, algorithmic composition and experimentation on Artificial Intelligence (AI) applied to music. Recent development of IT and its increasing availability also provides composers with even more creative possibilities to explore. The availability of the new cloud computing technologies also constitutes a valuable opportunity for the design of music computation architectures, pushing computer music into a new paradigm. Therefore, all the computing capabilities for creating music can be provided as services “without requiring detailed knowledge of the underlying technology” (Li et al., 2011).

This work is the result of several years of research in modeling music composition processes and music representation. The departure point for this research was the EvOntology (Alvaro et al., 2006), an in-depth study of the knowledge level (Newell, 1982) for music composition, which served as the basis for developing EvMusic (Alvaro et al., 2005), a productive composition environment written in Lisp programming language as a stand-alone application. Then, a remote graphic user interface was developed and published on the Web in a client-server approach. This was the first step towards a distributed version. The next step was to explore the possibilities that cloud computing could offer for different musical technologies to efficiently coexist in the same system. A suitable shared musical representation was found to be the key for distributing musical services in the new framework. Thus, the representation MusicJSON (Alvaro and Barros, 2010b) was designed on top of the reliable EvMusic representation. It was also optimized for network use by including a textual form, a database form, and an effective protocol for communication between music services. The next step was to conduct some experiments on the integration of musical services (Alvaro and Barros, 2010a), and their application to real musical composition (Alvaro and Barros, 2010b).

In this paper, the complete design and implementation of a cloud computing architecture for music composition is presented, incorporating the definition of a music-specific service layer called CMaaS. The proposal is the result of a long iterative research process. As far as we know, it is the first approach for a running distributed CM architecture for compositions based on
music services. It also features a high degree of music representation, flexibility, and a versatile framework for web application development. In addition, the paper includes two different applications on composition which validates the quality and efficacy of the implemented architecture model in very different contexts. The first one implements a complex model of symbolic composition with motives and harmonic constraints. The second ports the venerable Csound (Boulanger, 2000) to the cloud.

The paper is organized as follows. In the next section a short background on techniques and architectures being used in music composition is described. Section 3 presents the proposed cloud architecture for computer music. Section 4 is focused on giving details about the CMaaS layer and how domain-specific components are assembled in order to provide computer music services. Section 5 describes two working applications on composition, implemented at the Saas layer. The paper finishes with a discussion of the advantages of this type of architectures for creative activities and some conclusions about the results of this research.

2. Background

Music composition, understood as generating new music from rules (Delgado et al., 2009), has been the object of study and application in Computer Science and Artificial Intelligence (AI) during the last decades. Within this scope, some systems based on AI techniques have appeared to support the composition process. They include LISP programming (Taube, 1991), Case-based reasoning (Ribeiro et al., 2001), Genetic Algorithms (Moroni et al., 2000), restrictions (Henz et al., 1996; Anders et al., 2005), or ontologies and cognitive modeling (Alvaro et al., 2006). At first, all of them were stand-alone system. As information and communication technologies evolved, complex architectures were developed, such as Blackboard architecture (Laske, 1990), or multi-agent systems (Delgado et al., 2009).

With the WWW, client-server systems appeared (Manzolli and Verschure, 2005; Jordá and Wust, 2001; Bryan-Kinns, 2004; Alvaro et al., 2005; Manara et al., 2005; López et al., 2010; Wassermann et al., 2000) providing a single server-side application and thus avoiding installation in local computers. However, this setup had some disadvantages, like losing the whole control over every element of the system, and a limitation in the set of resources available in the server. The next generation is the cloud-computing architectures focused on music composition, whose presentation is one of the objectives of the present paper (Figs. 1 and 2).

As an evolution in IT, Cloud computing involves a new paradigm in which computer infrastructure and software are provided as a service (Ambrust et al., 2009). These services themselves have been referred to as Software as a Service (SaaS). Every day new applications appear in the Web, in the form of SaaS, providing the user with the capability of "working in the
cloud”, on which the information is no longer stored in local hard disks but in Web servers. Computation infrastructure is also offered as a service (IaaS), thus enabling the user to run the customer software. Several providers currently offer resizable computing capacity as a public cloud, such as the Amazon Elastic Compute Cloud (EC2, 2010) and the Google AppEngine (2010). With the introduction of virtualization technologies, infrastructure services have been expanded offering also virtual machines as services. A whole layer structure of cloud computing services has been proposed by Jones (2009) and Furht (2010) as shown in Fig. 3A. Two new layers have been introduced: one at the bottom corresponding to physical machines (dIaaS) and an intermediate layer corresponding to virtual machines equipped with an operative system (PaaS).

The systems based on cloud computing architectures benefit from the advantages of virtualization, distributed computing and high-speed network technologies. Scalable resources, load balancing, and availability on demand are also available for building more efficient systems. CM composition systems can benefit from this powerful technological environment. In addition, complex composition systems can be split into operational units specialized in a particular service. Such specialization is the key of this proposal. Thus the complexity of music composition systems is distributed among specialized services which can be shared by other systems. In the following sections, a pioneer and unique service-oriented cloud computing architecture for music composition is described.

3. A cloud architecture for computer music

Following the layers redefined by Furht (2010), this paper introduces a dedicated layer at the top of the PaaS layer: Computer music as a Service layer, as shown in Fig. 3B. After an iterative process of creating and defining CM experiments for applied music composition in the cloud (Alvaro and Barros, 2010a, b), all music services involved in these composition experiments have been standardized and offered as building blocks at this platform level.

Our approach is based on the assumption of the availability of these Music Services for building different CMC Systems. Therefore, the design of new music applications (at the SaaS layer) is facilitated by the joint work of different music services controlled by a web application. The key factor for a successful integration of the services lies in the use of a well-defined, suitable music representation for music data exchange in the cloud. MusicJSON (Alvaro and Barros, 2010b) which is the outcome of our research into CM Composition has been designed for this purpose. All the components in the CMaaS layer, as well as some examples of CMC integration in the SaaS layer are described below in a bottom to top approach.

The proposed architecture for Computer Music Cloud (CMC) is shown in Fig. 4. As described by Furht, the base layer of the architecture or dIaaS consists of the real hardware machines. It provides the computation power and the storage services demanded by the cloud.

3.1. IaaS and virtualization

As in many other clouds, the use and allocation of the hardware is optimized by the use of virtualization. This layer provides Infrastructure as a Service in the form of virtual machines. KVM (Kivity et al., 2007) is chosen as the virtualization engine due to its proven stability, flexibility, real hardware virtualization and open-source availability. Since it is not the aim of this paper to go into detail about the well-known features of KVM, some relevant sources are included for reference.

The aim of our implementation was not a final deployment of stable services in the web, but building some versatile architectures for the research and development of computer music composition in the cloud. If the context were to provide fixed services, a general purpose cloud manager like the latest OpenNebula (Fontan et al., 2008) could be used for this layer. However, within our research context, the required hypervisor must support our slot-based model (for this IaaS layer), the template hierarchy of images (at PaaS layer), and the sharing of source code among the services (of the CmmaS layer), while satisfying security restrictions of our computer lab (dIaaS layer). In order to satisfy such flexibility, a dedicated and lighter cloud hypervisor called cloudimage was developed.

The purpose of cloudimage is to manage images and virtual machines (VM), and assign them to virtualization slots. Cloudimage provides a proper management of images templates in the image hierarchy approach, as well as facilitates experimentation in the scalability of services. It also manages the restart of every active VM during the boot process of the host, as well as the clean shutdown of VMs during the shutdown. It was written as a bash script with some basic commands for starting, stopping, listing, and saving the state of VMs. The start command assigns a Service Image to a specific VM slot with some hardware parameters like memory assignment. The network connection of the VMs is simplified by using the port-mapping feature included in KVM. The list command prints a table of active slots, showing their status, KVM PIDs, and the parameters of their VM instances and images. As their names suggest, the stop command requests a clean shutdown of the VM in the slot, and savestate sends the savevm command to the KVM monitor for saving the state of the VM to a file.

The structure of this IaaS layer is shown in Fig. 4. Basically, it supplies VM slots to be inhabited by the Images of the PaaS layer. A cloud share is also available for every slot, for development purposes and the unification of the service code. This internal share is served by the host as a NFS (Network File System) and mounted by the service images when needed, exclusively for development and maintenance. As an example, by using this valuable feature, services can share the same source code of the libraries that they have in common, thereby facilitating maintenance. Although this sharing is available at all times, its functional use by the services should be avoided in order to preserve the operational independence of the upper layers. If, for
example this was used to share data, the image of the service would not work with other hypervisors, and therefore would not be reusable. For sharing data among the services, some dedicated music storage services are provided, as explained in Section 4, allowing the sharing of music data not only across the cloud architecture, but also across the entire WWW.

3.2. PaaS: Image standardization

The next layer in this cloud architecture provides Platform as a Service in the form of running service image templates. As shown in Fig. 4, the purpose of this layer is the allocation of Service Images from a Library into the VM slots provided by the IaaS layer.

In this architecture, all Service Images are standardized into the software structure shown in Fig. 5, with the operating system at the bottom. In this proposal, all services run Ubuntu Linux as the OS. Over this layer lie both the software libraries and the Interpreter/Compiler for the programming language of the service. One step above, there is a HTTP server as a requirement for the Service to be integrated into the CMC in order to accept requests from other services. Next to it, the service program code itself is represented. The service is also provided with an HTTP client, thus allowing it to make requests to other services. In this structure both server and client are written in the same Programming Language of the service program to facilitate integration.

In order to complete the communication requirements, the HTTP transport must also support JavaScript Object Notation (JSON) (Crockford, 2011) in terms of parsing and serialization. This feature will support the integration into the CMC because MusicJSON (Alvaro and Barros, 2010b), the text form of the music representation choosen, and its data exchange format, comply with JSON, as described in Section 4.3.

In this proposal, the Qemu Copy On Write (qcow2) (McLoughlin, 2008) image format has been chosen with the aim of image development simplification, and re-utilization of images. Among other features, the qcow2 format provides small image size and cow layering. That is, one image can be used as the base for a new image and only the modifications are written in the new image file. This valuable property not only considerably reduces the size of the images, but it also allows the organization of the service images in a “is-a” relationship-based hierarchy.

Fig. 4. Proposed cloud computing architecture for the computer music cloud. Each layer is detailed showing its elements.
Figure 6 shows the initial hierarchy of the service image templates in this architecture. The basic image template is a simple installation of the OS. For this approach, Ubuntu 10.4 has been chosen. Next, a new template is inherited and new installations are carried out in order to adapt the new image to the cloud architecture. The following packages were installed, among others: acpi for a clean shutdown of the VM, ssh access, a nfs client for using the development share, subversion for code version control, and emacs as code editor.

The next step in the template hierarchy has to do with the language to be used for programming the service. Three templates are provided: a LISP-based service, a Python-based service and a general purpose file + cgi server based on lighttpd (Kneschke, 2007).

The LISP service template uses Steel Bank Common Lisp (SBCL) as the programming language. Hunchentoot (Weitz, 2007), a robust http server written in LISP, was chosen as the required HTTP server. Drakma (Weitz, 2008) and Yason (Hubner, 2008) were also chosen as the LISP libraries for HTTP client and JSON management, respectively. For development and debugging of the service code, an IDE is provided consisting of emacs (Stallman, 1981) with SLIME (Mardsen, 2003) from a remote ssh connection to the image.

The Python service template uses the web2py framework (DiPierro, 2009), which includes a http server with database integration, a http client, and a JSON contributed library, among others. For code writing both emacs and the included editor are available.
4. CMaaS

The described PaaS layer provides, on demand, running image templates with the basic elements to build the services. In our approach, the SaaS layer of the applications does not rest immediately over the PaaS layer. The CM applications provided as a service consist of the aggregation of several music services working together. Hence, a new layer is introduced in between, as an extension of PaaS, incorporating all music services as the building blocks for the CMC in the SaaS layer. In this section, the model for a CMC is explained, followed by the description of the components of this Computer Music as a Service layer, as shown in Fig. 4, the Computer Music Reusable Cloud Services, the necessary music libraries which they are based on, and the framework for the development of CM web applications.

4.1. CMC model

The CMC model for the applications to be served at the SaaS layer is represented in Fig. 8. The computer music cloud application is a combination of several music services working together in a composition process. Music consists of symbolic music objects of different qualities and multiple abstraction levels (Alvaro et al., 2006). All these symbolic music objects taking part in the composition are stored in a composition environment which takes the form of a storage service. The composition process is carried out in this storage acting as the music sheet where the score is written. For a better understanding of the composition model, Fig. 7 shows a minimum set of CM services in operation. In its simplest form, a web application allows the user to remotely create and edit different kinds of musical objects stored in the environment service. These symbolic objects could be, at a low abstraction level, imagined as individual musical notes depicting the music composition inside the composition environment storage. An output service is provided for translating the stored music representation of the composition into a human-level music format like audio or standard graphic notation. This output format is fed back to the user and displayed in a dedicated window inside the web application.

A more complex CMC model is shown in Fig. 8, where several services have been included. In the example above, simple music notes have been considered for the sake of simplification. However, music composition combines music symbols at multiple levels of abstraction (Alvaro et al., 2006). For instance, composers often deal with music motives, chord progressions, composition engines, and many other high-abstraction objects (Alvaro and Barros, 2010b). As it is not the purpose of this paper to get deeper into those symbolic music entities, some references have been included. For the sake of understanding the model, it is enough to consider that high-abstraction music elements can be developed (i.e., expanded) into lower-abstraction ones. In Fig. 8, a new type of service, generally called music developer, has been included for this purpose. Thanks to this type of service, the composer can focus on more creative levels and deal with high-abstraction music objects in his composition, letting the music developer service do the tedious work. This way of composing is widely known as algorithmic composition (Nierhaus, 2010). As a particular case of this type of service, music generators are also represented. Music generators can be understood as developers using any kind of data as input, or even just a few parameters.

Completing the set of intelligent services of the model, Fig. 8 incorporates the music agent type. As described in (Russell-Norvig, 2002), an agent is a device provided with the capabilities for both acquiring information from the environment, and acting into it in an autonomous way. In this CMC model, the music agent analyses the composition stored in the composition environment service, and introduces changes and suggestions for improving the composition.

Music storage has been completed by providing two more types of services. The music library service stores standard music objects belonging to the shared music knowledge. It includes elements of the music language, such as the standard chord library, the family of music scales, and others. It can also include dedicated elements from other specific music languages. For example, in the application described below the shared music library service provides some Csound elements, like instruments, tables and mixers, to be used in the composition.

The musical data belonging to the user is stored in the user library for two purposes. On the one hand, it stores reusable music objects created by the user, such as musical motives, user-defined library objects, music engines and even entire music sections. The service makes all these music objects available for future reuse in any other composition. On the other hand, it stores the complete scores, as work projects, making them available at any time.

4.2. CM cloud services

As stated above, these services are the blocks for building computer music applications in the cloud. They really constitute...
the components served by this CMaaS layer. CMC services were classified by their role in the composition process model, which has been explained above, into five possible categories (Alvaro and Barros, 2010b): input, agents, storage, development and output. Development and agent types are considered intelligent services in the sense that they usually incorporate techniques associated with AI. Most types of service have been discussed in the explanation of the composition model in the previous subsection. The aim of input services is to incorporate and edit new musical objects, hence that the EvEditor framework has been provided at this layer to simplify the creation of new editors, as explained below. Figure 9 shows the rest of CMC service types differentiating storage services into Composition Environments and Libraries. As an example of integration, Figures 12 and 14 show how these service types will conform to two different CMC Applications for music composition: The Diatonic Composer and Csound meets the Cloud, both described in detail in Section 5.

The classification of CMC services facilitates the template and reutilization mechanisms of the approach. It is important to remember that, as explained in the PaaS section, services are implemented in a hierarchical approach by using image templates, that is, by customizing more general predefined services. This facilitates the definition of new services as subclasses to suit the new services requirements. As an example, a template for output services is provided. This output template incorporates the CMC integration requirements (MusicJSON protocol and representation), and the basic tools for the conversion. By customizing the conversion script of the template and including the required libraries, different output services can be easily implemented for the desired format. Some examples of customization can be seen in Fig. 12 including MIDI, PDF notation, PNG notation and MusicJSON formats. Other audio-related output services such as OGG, PNG Spectrogram, and CSD are shown in Fig. 14. Each instance of these services can attend multiple requests from different clients. Therefore it can even be integrated in several CMC at the same time. As a working example the same MIDI output service has been integrated in some CMC applications. This highlights one of the main advantages of our approach: once a functionality is implemented in a specialized CMC service, it can serve any CMC that needs such functionality. Consequently, once a library of specialized services has been made available by the CMaaS layer, you can implement a CMC application by bringing together the functional services needed. If a demanded functionality is not available, a new specialized service could be created by customizing a template, and then added to the library for future CMCs.

4.3. Music representation

The key for a successful integration of all these music services is a suitable and well-defined music representation for music composition optimized for the cloud. They all must share the same music language and understand what type of entities they are managing. At the same time, such representation must be optimized for music data sharing and transferring among the services. During our research, a complete music representation for composition has been refined as a result of many years of real music composition with computers. This representation includes both music entities and composition procedures, and it has been materialized into the EvMusic representation and composition environment. The ontological core of this music representation is based on the EvMetamodel (Alvaro et al., 2006) which provides
modeling of any time-based structure by a multi-level representation. Besides time, the second important dimension of music is pitch. A comprehensive survey has been carried out also on it, from sound frequency to complex musical pitch structures, resulting in SymbolicPitch, a coherent model for representing musical pitch that has been extensively proven in many music compositions. This mature musical performance ensures suitability for all kinds of musical elements involved in the process of composition. To adapt the system to the cloud, the representation has been optimized for the Web, providing it with both database and textual forms (Alvaro and Barros, 2010b). The database form is designed for the storage of musical objects in a database. It allows several CMC services to share the same musical data and collaborate in the composition process.

The textual form, called MusicJSON, allows music information to be interchanged among services through web streams, and to be stored as a text file. This format is basically JSON (Crockford, 2011) applied to the musical entities of EvMusic, ensuring the adequate representation of all types of musical elements involved in the process of composition. MusicJSON is not limited to serializing musical instances and their relationships in the form of JavaScript objects, it also incorporates a number of features designed specifically for its use within Web environments. On the one hand, MusicJSON handles extended references in the same way as links in a hypertext, which is especially important for referencing elements of a music library. On the other hand, it also includes a basic communication protocol for data interchange.

For more details about MusicJSON as well as some examples of use including the representation of musical scores, the use of extended references, and the file format, we recommend consulting the reference (Alvaro and Barros, 2010b). However, in the context of this paper, a simple example of usage as a protocol of communication within the architecture of this proposal is explained below. In this example, an editor service requests the storage service to modify the duration value of a musical note. The storage service responds to standard GET requests with a URL that ends with the function to perform, usually a CRUD function (Create, Retrieve, Update, Delete). The request is accompanied by two parameters: a data parameter with information in MusicJSON format and a second parameter named callback, as the JavaScript function to be returned. In this example the URL is http://evmusic.fauno.org/storage03/update, the callback parameter is stcCallback1002 and the data parameter is (“duration”:12, “ref”:“note_84”), requesting to change the duration of the note with ref “note_84” to a new value of 12.

After updating the corresponding data in the storage service, the returned response has the following form:

Callback ("message": "Message Content", "data":[response data], "success": true)

This means that the returned data are sent back along with a status message confirming the transaction, everything encapsulated in a function with the name of the callback, as required by the JSONP data transaction, so the service can accept AJAX requests from other domains. In our example, here is the response received to the update request above:

stcCallback1002 ("message": "Updated record", "data": {"track": 1, "objclass": "note", "pitch":39, "start": 6, "duration": 12, "ref": "note_84", "id": 84}, "success": true)

The communication protocol between any of the CMC service follows this model of interchange and modification of objects. In this example the transaction object corresponds to a musical entity, but the protocol is open to interchange any object, like those more specific to the coordination of processes, such as tasks, queues and requests. By using MusicJSON as representation of musical entities and as exchange protocol, it is easy to integrate any music service within an operational computer music cloud.

4.4. CM libraries

Most of the music services described above are built upon music libraries based on EvMusic and MusicJSON (Alvaro and Barros, 2010b). They incorporate the implementations of composition models, abstract music entities, algorithmic composition engines, and other intelligent music libraries. Some other music services use third-party libraries such us Lillypond (Nienhuys and Jan, 2003) for graphic rendering of scores, or Csound (Boulanger, 2000) as a sound synthesis engine. The set of these structural elements and representation in which they are implemented constitute the essential musical substrate of this CMAaaS layer.

In order to facilitate the development of musical Web application for the user, EvEditor (Alvaro and Barros, 2010b), a versatile graphic environment for musical objects edition has been developed as a framework.

The Web application runs at the client side, inside the standard Web browser. That is why JavaScript (ECMAScript) became the application language and the ExtJS library (Sencha) was chosen as the base for development. ExtJS is a powerful JavaScript library including a vast collection of components for user interface creation. Elements are defined in a hierarchic class system, which offers a great solution for our Object-Oriented Programming (OOP) approach. That is, all EvEditor components are sub-classed from ExtJS classes.

The web application takes the shape of a desktop with windows metaphor, a well-tested and intuitive interface environment, but within a web-browser window.

As shown in Fig. 10, EvEditor client is a web application consisting of three layers: data, component and DOM. The layer at the bottom, or data layer, contains a storage proxy for the data that is being edited. Figure 11 details the proxy-based edition mechanism at this bottom layer. The left side of Fig. 11 shows the Data Edition scheme used by EvEditor. The records to be edited in the remote storage service are duplicated in a proxy. The database in the remote storage service is synced with the editions and updates that the editor writes in its proxy. When the confirmation of the data update is received, the graphic update is performed in the editor. Several editors can share the same proxy, so every listening editor is updated. The same proxy approach is used for the communication between other CMC services. The right side of Fig. 11 represents the Service Request transaction, based on the same mechanism, between a Web application interface and an output CMC service. A request from the interface modifies a service transaction record in the service proxy and sets its status to wait. The proxy performs the real service request by updating the internal record of a task dispatcher inside the remote CMC service. In the cloud model, the mission of this dispatcher is to assign the requested task to an available service instance and to update the record accordingly. Every change in the request record is replicated by the proxy in the application, and every element of the application interface which is listening to this proxy reacts showing the service response.

The intermediate layer shown in Fig. 10 is a symbolic zone for all interface components. It includes graphic interface components for CMC services, editor windows, container views and robjects: representations for the musical objects under current edition. All interface components are sub-classed from ExtJS components. Every editor is displayed in its own window on the
working desktop and optionally contains a contentView displaying its child objects as robjects. In the third layer, or DOM (Document Object Model) (Wood, 1999), all interface components are rendered. Figures 13 and 15 are browser-window screen captures showing the working desktop and some editor windows. The application menu is shown in the lower left-hand corner. It includes the user settings. The development of EvEditor provides this CMaaS layer with a versatile framework for the development of general-purpose object editors under an OOP approach. Thereby, specific music editors are created as subclasses with the customizations required by the music object type.

5. SaaS: Music composition applications in the cloud

Having reached this point in the description of the proposed architecture, CMaaS layer has made available a range of musical services and a framework for building Web client applications. The next step, performed by the upper layer called SaaS, is to build applications for musical composition in the cloud, by gathering various music services and a web application containing the user interfaces, as shown at the top of Fig. 4. The component services are obtained through customization of the templates served by lower layers. In terms of design and operation, this approach provides several advantages over the traditional one. From a usage perspective, it provides all the benefits of the applications in the cloud, such as operation from any browser connected to the network, or storage of work projects on the Web. From an implementation viewpoint, it is no longer necessary for a single system to include, in a single platform, all the services involved in the process of musical composition. This platform independence diminishes the problems of usability and aging of services. By contrast, each service component of the CMC can be now designed to provide the best performance in the platform that is most appropriate for its implementation, regardless of the platform chosen for the rest of the services. At the same time, each of these specialized services can be reused as part of other CMC applications. This approach also permits easy integration of existing music services, installing them in a service image and providing them with the required accommodations to the new musical representation MusicJSON and its communication protocol. As an example, the integration of the Csound sound synthesis engine (Boulanger, 2000) in a CMC application is described in Section 5.2.

Fig. 10. Layer structure of EvEditor.
In the following subsections we present two examples of CMC working applications, each of them focusing on one of the two main areas of computer composition: score and sound. The first application, called “Diatonic Composer”, is based on our previous works (Alvaro et al., 2005; Alvaro and Barros, 2010a, b). It focuses on the traditional conception of music composition producing a standard notated score, but instead of writing every single music note, the creative process is carried out at higher levels of abstraction, i.e., closer to the mental idea of the composer. The second application, called “Csound meets the Cloud” is presented as a novelty in this paper, and it is focused on sound synthesis, but again, composing from higher levels of abstraction.

5.1. Diatonic composer

As a first example of integration of music services to conform to a CMC, the application “Diatonic Composer” implements the following composition model.

5.1.1. Composition model

The musical piece is created by combining diatonic musical object. It is similar to the way a painter creates his work by distributing different thematic elements over his canvas. This method of composition was described as motivic composition by A. Schoenberg in “The Gedanke manuscripts” (Schoenberg, 1995), where the concept of motive is widely discussed. For Schoenberg, a motive is the minimum musical unit with self-identity, that is, a thematic entity that can be recognized each time it appears, despite the changes that may have occurred. Musically, these relationships of similarity between the structural elements provide the composition with form coherence. At the level of music representation, it is important to notice that motive is really a musical entity belonging to a higher level of abstraction, as it is formed by a group of notes, a main construction element of lower order.

This procedure of composition with motives has been extended and improved through the introduction of the diatonic
In Schoenberg’s model, the pitches of the notes which form the motive are absolute. However, in our diatonic model, the pitch of the notes is always considered symbolic, and its final value is determined by a number of restrictions imposed by the harmony of the moment in time. In this sense, harmony is modeled as a vertical schema of constraints which is applied to
The whole section at each vertical slice of time. This unique composition model based on symbolic pitch, harmonic constraints, and motivic segments greatly enriches the absolute pitch approach. This model has been already implemented in EvMusic and successfully applied in many chamber music and orchestral works performed internationally.

In the paradigm shift discussed in this article, the application “Diatonic Composer” moves the described composition model to the Cloud. Figure 12 shows the integration map of the services involved in this application. Since the basic model of CMC was already described in Section 4.1, only the particular customizations for the application are discussed now.

The main customizations introduced in the model consist of the music data types involved in the composition model, their correspondent editors and library elements, the output services, and the developers needed for the high-level music objects.

5.1.2. Music objects and services

All music objects involved in this composition model are arranged and edited inside the editor windows of the Web application. Most of them are created from scratch, while others are copies of previously defined instances stored in library services. The use of the library is facilitated with drag and drop actions.

The main music composition unit is the diatonic section, whose editor is shown in the center of Fig. 13. In the main area, all sub-events of the section are disposed on a pitch vs. time canvas. Thus, each sub-event can be placed and dragged across both time and pitch. Three types of events are shown: the single diatonic note, the part or group of sub-events identified by a two-note icon, and the motive segment, identified by puzzle piece icon. All sub-events are organized in tracks identified by color, and they can be edited in detail by opening a dedicated editor window with a double-click. The top area of the window contains the vertical constraints to be applied to the diatonic section. They include the harmonic constraints, such as the chord and the scale changes.

The development of the diatonic section is carried out by the developer service shown at the top of Fig. 12. It performs the algorithmic development of every sub-event according to its type, and solves their pitch values by applying the harmonic constraints. As a result of the development, a plain MusicJSON score format has been made available.

The output of the developer can then be passed to the output services. In this CMC, four output services are included. The MIDI service provides a sonic representation of the section to be reproduced by the browser. The PNG service renders a graphic standard notation of the music clip to be shown in its service window. The PDF service renders a more detailed graphic score ready to be printed. Both PNG and PDF services make use of the Lilypond (Niemhuys, 2003) library. The fourth output service serializes the score as MusicJSON notation format for download.

5.2. CSOUND Meets the cloud

This CMC application focuses on music composition with sound synthesis. Csound (Boulanger, 2000) has been chosen as the synthesis engine for the application. Csound is probably one of the most recognized languages for sound synthesis. Its robustness and its availability for many platforms make it the right choice. The following application develops an expandable synthesis-based composition environment as a foreground for Csound placed in the Cloud.

5.2.1. Composition model

As in the previous model, the music piece is composed by arranging music elements graphically on a pitch vs. time canvas. In this case, the target is not the graphic score, but the synthesized audio, and the building blocks to be spread over the canvas are new entities called sonic elements. The transformation from a music canvas with distributed sonic elements into audio is performed in several steps. First, every intervening sonic element has to be developed into basic MusicJSON sonic-events. Then, the MusicJSON sonic-section is transformed into .csd; the input format for the Csound engine. The audio file is then synthesized by Csound, and then compressed into .ogg audio format for an optimized network transmission. Finally, the .ogg file is played back by the browser.

From the viewpoint of music representation, two points are noteworthy here: the use of the sonic-aggregate entity as the main composing element and the MusicJSON sonic abstraction. As stated, the composition is generated by populating the canvas or sonic-section with these sonic-aggregates. They are meta-events representing sets of related sounds scattered around the bounded area in the canvas. They could be recognized as sonic clouds, swarms or pointillist textures, as well as time-structured distributions of evolving sonic-events. This type of meta-event is implemented as a subclass of the sample-generator, a powerful algorithmic engine included in the EvMetamodel (Alvaro et al., 2006). It represents the set of child events by defining the evolution of each of its parameters and a time-sampler function. During the development of the meta-event, it generates at every sampling time one child-event with parameters whose value is a sample from the evolving value of its respective parameter. Thus, complex distributions of evolving events can be easily defined. This feature, along with the expressive power of parametric sound synthesis, has great creative potential.

5.2.2. Music objects

In the composition model of Csound, there are two components to be defined: the score and the orchestra. The sound will be produced by assigning every parameterized events from the score to its corresponding synthesis unit or instrument of the orchestra. Bearing this in mind, a sonic abstraction in the proposed model is introduced at the MusicJSON representation layer. In the orchestra component, this abstraction improves the usability and the definition of instruments by separating the sound generation part (sonic-instruments) from the mixing part.
(sonic-mixer), and storing the normalized sonic-instruments in a library service. Thus, the same instrument engine can be used in different pieces. This also solves the common requirement of a remix into a different number of channels, by changing just the mixer part.

In the score component, some sound parameters of the sonic-event such as sound pitch or sound intensity are normalized by the MusicJSON sonic abstraction. Also, mixing parameters have been standardized into the positioning parameters: distance, pan-angle, elevation-angle, and rev-amount. This completes the instrument usability, and also facilitates its integration with the EvEditor system. Waveforms and tables are also managed by the library service as entities of the sonic-table type.

5.2.3. Music services

Figure 14 shows the service integration of the CMC application. All services are grouped by type for clarification. As in the previous application, storage services include the composition environment, where the composition is taking place, and the music and user libraries. In this case, the libraries store the dedicated music objects mentioned above, such as sonic-instruments, sonic-mixers, and sonic-tables. User library also includes the user’s favorite sonic-aggregates to be reused, and even some sonic-sections as examples for future works.

Three output services are included: the audio output in the .ogg format for listening, the graphic spectrogram generator for a visual inspection of the generated audio spectrum, and a .csd file exporter for the purpose of archiving. At the top of Fig. 14, two services of type intelligent, based on AI techniques are represented. The sampler-generator developer is an algorithmic composer that performs the development process described above. It generates child-events from a sonic-aggregate event. Another notable feature of this CMC is the introduction of a distribution agent in the composition process. This agent is the second intelligent service incorporated in this CMC. Its mission is to supervise the distribution of sonic-aggregates and adjust their position both in time and pitch dimensions. In time, it trims the event positions (inside an allowed time margin) to minimize simultaneous condensations of sound and silences. In the pitch dimension, it shifts the pitch...
position of the events according to two rules: a density rule to minimize the event density at low pitches, and a harmonic rule to favor “natural” frequency relations among simultaneous events.

Figure 15 is a screen capture of the “Sound Meets the Cloud” web application. In the center, a sonic-section is shown with some sonic-aggregates and sonic-events, represented by colored rectangles with an identity icon. At the top left there is the library window. On the right, the player of the .ogg output service is shown. For the sake of illustration, in the lower part of the screen, a network inspector shows the response of the service in MusicJSON format to a simple edition request.

6. Conclusion

This paper presents a working cloud computing architecture optimized for computer music. Based on the cloud layer structure proposed by Furht (2010), a detailed description of each layer is carried out from bottom to top, ending with the description of two working music composition applications. At the infrastructure layer (IaaS), the virtualization techniques employed in the architecture have been described. At the platform layer (PaaS), a standardization of service images based on a hierarchical template-based approach is contributed, thus facilitating the creation of new, more specific images by customizing other, previously-defined ones.

The main contribution of this paper is the introduction of a new dedicated layer just above PaaS. This new layer is called Computer Music as a Service or CMaaS. The mission of this layer is to provide functional and specialized computer music services as the building blocks for assembling computer music applications at the SaaS layer. Apart from the intrinsic benefits of the cloud, this new approach offers three main advantages over the traditional CM composition systems: distribution, specialization, and reuse.

As a distributed system, it is no longer necessary for the system to include in a single platform all the services involved in the process of music composition. One platform can be more suitable for one service than for others. The goal is now accomplished by distributing tasks into multiple platforms, even running in parallel. The distribution also makes it easier to create applications by connecting available building blocks. The key to a successful integration of the services into a computer music cloud for composition is the selection of a suitable music representation. Therefore, the approach uses MusicJSON (Alvaro and Barros, 2010b), a cloud-oriented music representation that incorporates a music data exchange protocol.

As a specialized integration, every CM service can be tailored to provide the best performance and implemented in the most appropriate platform. The application benefits from this specialization with increased performance.

The re-utilization feature of this service-oriented approach implies that each of these specialized services can be reused as part of a different CMC. This is particularly useful in a research environment, where a rapid prototyping of testing applications is frequently required. The availability of services can be exploited into a scalable approach, where new instances of the same service can be incorporated on demand.

The CMaaS layer also provides tools for building the web application, acting as the user interface and manager of the CMC. In this sense, EvoEditor is provided as a versatile framework for the development of general-purpose object editors under an OOP approach.

At the SaaS layer, the implementation of two computer music applications for composition is described. “Diatonic Composer” contributes a productive composition model for scores, based on the visual arrangement of high-abstraction music elements such as diatonic motive-segments inside a pitch vs. time canvas with controlled harmonic constraints. The second application described in this paper, “Sound Meets the Cloud”, implements a high-abstraction composition model focused on sound synthesis. Some of its musical contributions are a flexible sonic representation in MusicJSON, a highly-efficient, parameter-based algorithmic developer, and the integration of a music agent in the composition process.

Beyond their specific contributions to music composition, “Diatonic Composer” and “Sound Meets the Cloud” applications are two working examples of the integration of several types of computer music services, provided by a CMaaS layer on top of the whole cloud computing architecture. They are a step forward in the bringing of computer music to the new distributed computation environment for Web music applications.

The modeling of a music composition environment by a storage service with a suitable music representation is also a key contribution of this approach. It sets the basis for sharing the same music composition among several users and agents. It thus opens the door for future work in collaborative composition, music tutoring applications, and music artificial intelligence systems consisting of specialized agents.

The architecture described in this paper also constitutes an interesting novelty, since, as far as we are aware, it is the first service-oriented cloud computing architecture for music composition.

Acknowledgements

This work has been partially supported by the PATIO project (patio.lcc.uma.es) (TIC-4273) of the Junta de Andalucía, Spain.

References


