2010 24th IEEE International Conference on Advanced Information Networking and Applications

A DSL-based Approach to Software Development and Deployment on Cloud

Krzysztof Sledziewski[1], Behzad Bordbar[1] and Rachid Anane[2]
School of Computer Science, University of Birmingham
{ k.sledziewski, b.bordbar}@cs.bham.ac.uk
[2]Faculty of Engineering and Computing, Coventry University
r.anane@coventry.ac.uk

***Abstract*— With the advent of Cloud computing massively scalable and cost effective IT resources can be accessed and used seamlessly. Various APIs are made available for manipulating the infrastructure of the Cloud and its data models and for applying the deployment tools. Cloud computing promotes a new approach to software development. In particular, the development team must bridge the gap between the requirement of the clients and the available facilities on the Cloud. This complexity might inevitably result in higher cost and potentially unsatisfactory results.**

In this paper a method for bridging the gap between the clients view and software development on the Cloud is proposed. It is based on the introduction of Domain Specific Languages (DSL) into the process of Cloud based application development and deployment. Domain Specific Languages facilitate the development of applications by easing the design of high level models and specifications that the client can understand and even produce. The automated method described in the paper implements and deploys software for the Cloud. A preliminary evaluation shows that the proposed approach improves the process of developing and deploying applications on the Cloud.

***Index Terms*— Cloud Computing, Service oriented Architecture, Domain Specific Languages, and Google App Engine**

I. INTRODUCTION

Cloud Computing provides a novel approach for software deployment and for hosting IT systems on the Internet, by managing the resources in order to enhance reliability and scalability. The cost of utilizing resources such as computing power, data transfer and data storage is relatively low since consumers only pay for the resources that they require [1][2][3][4].

Cloud computing adds however an extra layer of complexity into the process of software development. There is a steep learning curve associated with the APIs for interacting with the Cloud infrastructure, mastering new data models and software tools provided by Cloud vendors. Furthermore each Cloud provider introduces its own set of proprietary APIs data format and supporting software tools [5]. As a result, the path from the gathering of requirements to the deployment of the software on the Cloud is now being lengthened by the infrastructure created by the Cloud. Such complexity will not only extend the application development process it will also add to the cost of Cloud Computing. There is an urgent need to introduce better approaches to software development in order to exploit the potential of Cloud Computing.

The desire to reduce costs and speed up the software development process has also been marked by an increasing interest in Domain Specific Languages (DSL) [6][7][8]. In contrast to general-purpose languages such as C++ or Java, a Domain Specific Language is tightly coupled with a particular application domain and provides a higher level of abstraction. A Domain Specific Language hides low level details and allows the designer to focus mainly on the design. Moreover, DSLs use tools to automatically generate an implementation. As a result, software can be developed faster and with limited programming knowledge [6][9]. Finally, if the developer is familiar with the application domain, using DSLs require substantially less training [10]. Automated code generation in DSLs can reduce human induced errors and shorten the development cycle.

This paper presents an approach that incorporates Domain Specific Languages into the process of developing and deploying software on the Cloud. The proposed approach involves two fundamental steps. The first step concerns the development of a Domain Specific Language for a given domain so that the developers can generate high-level representation of an application. In the second step the language is used to generate automatically the code and to deploy it on the Cloud. The emphasis is on the seamless development and deployment of Cloud based applications. As a way of providing greater accessibility to the language by Cloud application developers, the DSL designed in the first step will be available in the form of Software as a Service (SaaS), which can be accessed via web browsers. This eliminates the need for additional DSL tools on a local machine.

The remainder of paper is organized as follows. Section II provides the background material on Cloud Computing, Domain Specific Languages and SaaS. Section III presents the rationale and focus of this research. Section IV describes the proposed approach for integrating Domain Specific Language into the development and deployment of Cloud based applications. Section V provides details of the implementation of the proposed approach. Section VI describes a case study where the approach is applied. Section

1550-445X/10 $26.00 © 2010 Crown Copyright
DOI 10.1109/AINA.2010.81

VII offers an evaluation of the proposed approach and Section VIII concludes the paper.

II. PRELIMINARIES

A. Cloud Computing

Cloud Computing [1][2][3][4] is a technology which introduces a novel way of accessing IT resources. It allows access to hardware, virtualised computational power, data storage, development and execution platforms or applications, in a service-oriented manner. These are characterized by being massively scalable and accessed through the Internet.

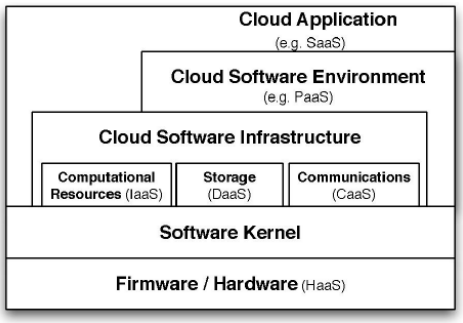

Figure 1: Cloud structure [11]

Cloud Computing offers services in many areas and the technology is often presented in the form of a layered architecture. For instance, Youseff et al. [11] present Cloud Computing in terms of five layers:

- Hardware Layer responsible for providing physical infrastructure for higher layers, such as hardware and networking systems.
- Software Kernel Layer which handles basic control over physical infrastructure including OS kernels, hypervisors, virtual machine monitors and clustering middleware.
- Cloud Software Infrastructure Layer is responsible for delivering computational power, data storage and communications infrastructure in the form of virtualised abstraction. The service-oriented manner of delivering these artefacts are usually called Infrastructure as a Service (IaaS), Data as a Service (DaaS) and Communication as a Service (CaaS).
- Cloud Software Environment Layer offers a run time environment for the applications that run on the top of Cloud's stack. It includes APIs, which allow applications to interact with the underlying Cloud infrastructure. The services provided by this layer are usually referred to as Platform as a Service (PaaS).
- Cloud Application Layer is the visible part of the cloud to end users, exposing services in the form of software applications which are commonly known as Software as a Service (SaaS). Such applications are usually based on web technology and are accessible by means of the Internet.

The following section provides a brief description of a specific Cloud Platform, the Google App Engine.

B. Google App Engine

Google App Engine is a platform, which facilitates the development and execution of software as a Service. The applications that are executed within the context of Google App Engine run on Google's massively scalable physical infrastructure [12]. The platform supports Java and Python runtime environments and provides a set of rich APIs, which serve as an intermediary between hosted application and virtualised infrastructure. The applications can store data in Google's scalable data store through a specific API. This data store differs from traditional relational data model. Instead of being held in tables, rows and columns, data is stored within entities and properties [12]. Google provides an SDK which simulates the Cloud infrastructure and runs on local machines in order to support the development of applications for the Google App Engine. A tool is also supplied for the deployment of applications on the Cloud [12].

C. Domain Specific Languages

The design of languages, which are specific to an application domain, is a common practice. For example, Specialised Querying Language (SQL) is specific to the domain of relational databases. Such languages allow problems to be modelled at a higher level of abstraction compared to general-purpose languages such as C++ or Java. The idea of using languages, which are specific to a domain, has been widely accepted and extended. Modern DSLs are produced and maintained by Software tools [6][7], which facilitate three main aspects:

- The design of a modelling language
- The generation of domain specific code in low level languages
- The provision of a framework for modelling and code generation for the developers.

The next section describes briefly one of the Software tools.

Microsoft DSL Tools

Although it is feasible to develop a Domain Specific Language from scratch, it is however a complex and time consuming process [8]. Microsoft has introduced a toolkit, the 'DSL Tools', which is widely used for the development of Visual Domain Specific Languages [7]. A domain model designer uses the toolkit to generate the language structure by developing a *metamodel* which captures the domain concepts and their relationships. The metamodel is specified in terms of UML class diagrams. Once the metamodel and the visualisation elements of the language have been generated, it is possible to produce code and in effect execute the DSL. The execution of the language is done within the context of Visual Studio. When the language is executed, a new Visual Studio instance is generated producing a Framework for the designer to use the language, for example by creating instances of the language in C# and executing them.

One of the components of Microsoft DSL Tools toolkit is the *generator*, which facilitates the traversal of the instances of the domain model and the generation of artifacts such as code and configuration files. In order to generate the output

from instances of the domain model, 'text-template files' have to be provided. Such files are made up of a dynamic part, which include C# code for specifying how to traverse models, and a static part that describes the output of the generated file.

D. *Software as a Service*

Software as a Service (SaaS) allows the execution of the software within a remote server, and its access by means of a standard web browser [13][14]. In contrast to packaged software, applications delivered in SaaS fashion are owned and hosted by the service provider. To be able to access applications, customers have to subscribe and pay for them [15]. The SaaS makes the process of deploying and upgrading software less complex and more cost effective [15]. The software is installed and maintained in one place only, in the hosting environment. Software maintenance is moved to the software provider side and upgrades are transparent to the users. SaaS model is platform independent and does not force users to rely on single software vendor. They can easily switch between different software providers [15].

III. PROBLEM STATEMENT

Although Cloud Computing presents an advantageous model for software execution, a wide range of technical expertise is required to build applications that run on a Cloud platform [5]. This covers knowledge of APIs that interact with the Cloud's infrastructure, new data models, tools for software development, local run-time environments simulating Cloud's infrastructure and tools that allow the deployment of applications on the target Cloud platform. Moreover, in the current market a wide range of Cloud platform is available. Each Cloud platform introduces different ways of accomplishing the same task. For instance, programming languages, APIs, data models or development tools may significantly differ from one Cloud platform to another. The skills acquired for specific platform, through training and experience, may not be transferred to another one [5].

Although Cloud Computing can accommodate the SaaS model, the inherent complexity of software development may be a serious impediment to its wider adoption. The challenge therefore is to ease the process of software development for Cloud by insulating the developer from the complexity and intricacies of the underlying models and tools.

IV. PROPOSED APPROACH

The aim of this work is to provide a framework for developing Domain Specific Languages, which facilitate the modelling and deployment of Cloud based applications. This involves using DSLs to create a seamless environment, which hides the Cloud implementation and deployment details, and enable the application designers to focus mainly on the problem domain. The process involves the following. Firstly, the DSL Developers produce a Domain Specific Language. Next, Application Designers use the language to model applications. The produced DSL is provided in the form of Software as a Service and target Cloud platform. An outline of the use of DSL by the Application Designer is depicted in Fig.2. Once the model is created by the Application Designer, the framework supporting the DSL, which is a SaaS, translates the model into executable Cloud specific code and automatically deploys it to the target Cloud platform. Users can access the deployed applications via browsers.

One of the key advantages of the presented method is that the Domain Specific Language allows the Application Designers to model applications using high level abstraction models for representing problem domains. In contrast to software development with a general-purpose language such as C++ or Java, implementation details are hidden from Application Designers. A combination of high-level modelling and automated code generation results in better design and shorter development cycles.

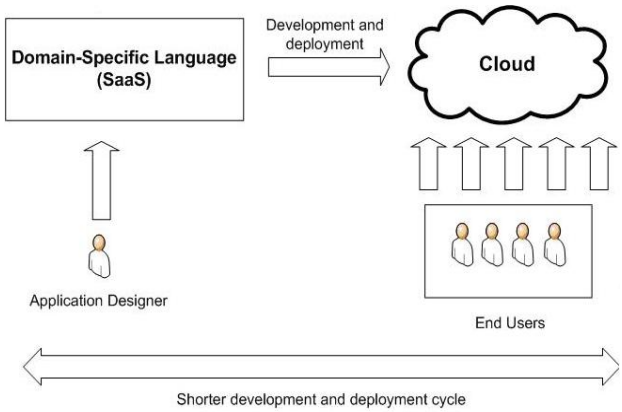

Figure 2: Diagram for the proposed approach

There are many reasons for presenting the DSL as a SaaS. Domain Specific Languages, which are delivered in the form of classical packaged software, suffer from poor accessibility [15]. The Application Designers need to install on their local machines several tools with which they are expected to be familiar. For instance, languages, which are built with the use of Microsoft DSL Tools, are executable within the context of Visual Studio [7]. The Application Designers need to install the development environment on their local machines and they have to be become familiar with its operation. To overcome this limitation, we introduce Domain Specific Languages provided in the form of Software as a Service, such that the Application Designers need only a standard web browser to access the language. The requirement for additional tools is moved to the side of the hosting environment.

V. OUTLINE OF THE IMPLEMENTATION

The development of Cloud applications with the method presented in this paper requires two main phases: the Development of the DSL and the provision of the DSL as a SaaS to be used by the application developer, as depicted in Fig. 2. In this section the two phases will be briefly explained.

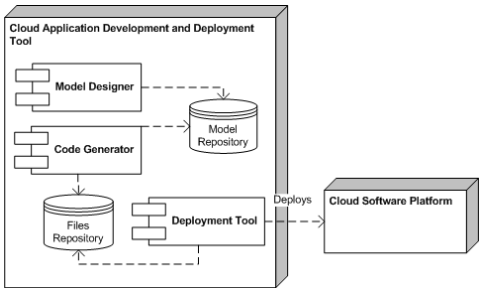

Figure 3: Architecture of the Tool

Phase 1 Development of the DSL: Development of Domain Specific Language is done by via Microsoft DSL Tools [7] to be executed within the context of Visual Studio. This phase involves two main steps:

1. Interaction with the domain experts to identify the Metamodel, a model capturing concepts involved in the domain and their relationship together. A model in the system is an instance of a Metamodel. Microsoft DSL Tools allows creating visual models, which represent each application.
2. The specification of templates for code generation from the models, instance of the metamodel. Specification of such templates allows Microsoft DSL Tools to automatically create C# code for executing the models.

In the conventional DSL approach, the Domain Specific Languages developed in this phase are executable within the context of Visual Studio, i.e. C# code can be generated and executed. For example, the template code generation, in step 2 above, can produce a two-tier system involving a database and an HTML form automatically. However, the method presented in this paper does not rely on the use of Microsoft DSL Tools by the Application Designer. More specifically, the DSL designer use the Microsoft DSL Tools, but the Application Designer uses a SaaS as depicted in Fig. 2. The structure of the SaaS, which reuses some of the components of the Microsoft DSL Tools, is explained in Phase 2.

Phase 2 Development of a SaaS: this will enable the Application Designer to use the DSL. The architecture of the SaaS is depicted in Figure 3. It comprises of three main components; these are Model Designer, Code Generator and Deployment Tool. Model Designer is a component, which presents the Domain Specific Language to Application Designers. It allows Application Designers to develop models representing instances of a domain problem. As the models created by Model Designer are used by further components, they are stored within the Model Repository. The Code Generator is used

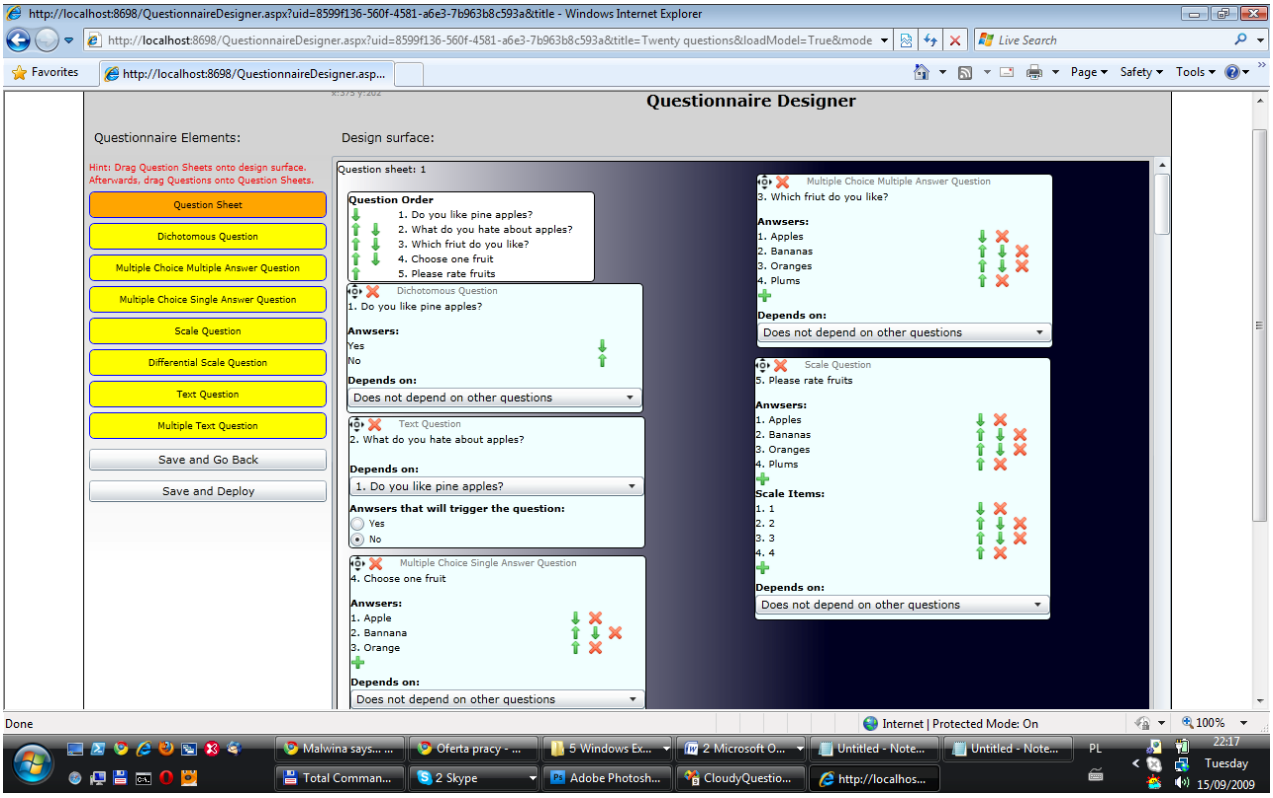

Figure 4: Screenshot of the Model Designer

to generate executable code from models created by the Model Designer. This component is a reused part of the Microsoft DSL Tools toolkit. The executable code, which is produced by the Code Generator, is stored in files that comprise Cloud based applications. The last component of the application is the Deployment Tool. This module is responsible for deploying the code produced by the Code Generator for the Cloud platform. It wraps the deployment tools delivered by the Cloud providers. When executed, it consumes files containing the generated application, executes the external deployment application obtained from Cloud's platform SDK and places the generated code as input for the deployment of the generated application on the target Cloud platform. The application is built with the use of ASP.NET platform.

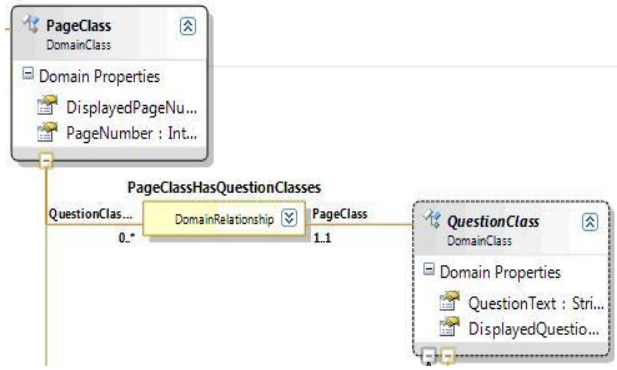

Figure 5: A metamodel section for the DSL

A case study based on the use of questionnaires for knowledge elicitation is used to illustrate the different steps of the framework. The domain knowledge for this application was obtained through an interview with a domain expert. An IT based questionnaire application is designed for gathering information online on a selected topic and for analysing the data provided by the participants. The online questionnaire is a web-based form, which presents different types of questions for the respondent to answer. There are various types of questions as explained in details in [16]:

• Dichotomous Questions, which include two answers and where only one of them can be valid.

• Multiple Choice that can have either a single answer or multiple answers.

• Scale Questions which are aimed at identifying the rate of some feature by using a fixed scale.

• Text Questions requiring answers in the form of one or more sections of text

The questions can be grouped into Question Sheets to improve readability and usability. As the application is online, the display should be in the form of HTML tables, where the columns correspond to the answers for the question, while the rows present the set of answers given by a certain questionnaire respondent.

The case study has resulted in a SaaS that allows the modelling of questionnaire applications, by the Application Designers, which automatically generates the executable code and deploys it to Google Cloud platform.

The Model Designer, see Figure 3, is a core component of the SaaS. The Model Designer is based on a Domain Specific Language, which allows the creation models of questionnaires in a visual manner. Figure presents a screenshot of the Model Designer, which is used by the developer to design questionnaires. To use the Model Designer, the Application Designer logs in to the SaaS, and if access is granted is redirected to the list of already developed questionnaire models. At this stage, the Application Designer can access an existing model or create a new one. The next step is a redirection to the Model Designer module.

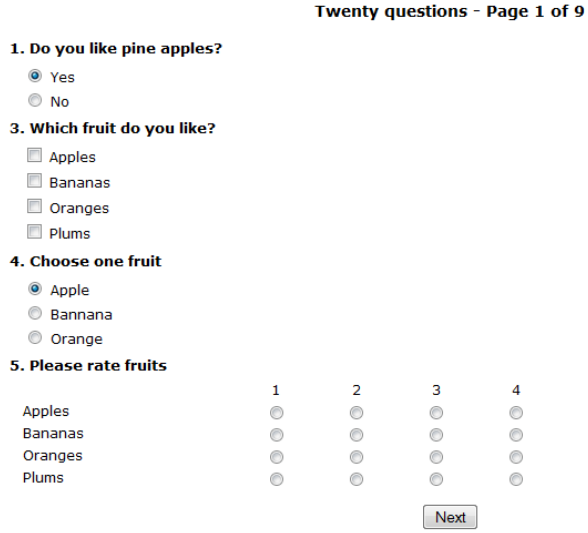

Figure 6: Sample questions

At the heart of the Designer Module is the DSL. The visual elements of the language correspond to the domain concepts relating to the design of the questionnaire. This correspondence is captured in a *metamodel*, a model itself, that describes the elements involved in the model of the application and relationships between these elements. Fig. 5 represents two of the classes involved in the metamodel of the DSL for the questionnaire. The class "PageClass" represents the page that contains a number of questions. Questions are instances of the "QuestionClass". Fig. 6 depicts an example of a page and a number of questions. Fig. 5 is an instance of a metamodel. For example, the page is an instance of "pageClass" element of the metamodel and each of the five questions is an instance of the "QuestionClass". Each question has a text, which is an instance of the attribute "QuestionText" of the metamodel which is of type "String". Each Page may include zero or more questions, which are captured as cardinatlities "0...*" and "1..1" in the "DomainRelationship."

Once the questionnaire model is built, the Application Designer is redirected to the Deployment tool (see Figure 3). The user provides his Cloud account credentials and deploys the questionnaire to the target Cloud platform. The current

tool is limited to deploying questionnaires to Google's Cloud only. If the deployment process is successful, a new instance of a web browser opens and the deployed application is made available. Finally, questionnaire respondents are able to access the deployed questionnaire application and fill in the questionnaires. They access the application by the means of web browser and URL given by the Application Designer. An instance of a deployed questionnaire application can be seen in Fig 6.

VII. EVALUATION

This section is concerned with the evaluation of the proposed method in terms of four criteria: amount of automatically generated code, time required for the development, maintainability and usability.

A. Amount of code

The first step of the evaluation process involves the investigation of the amount of code, which is automatically generated for different questionnaires. Four questionnaires were developed; these contain five, ten, twenty and thirty questions respectively.

Table 1 presents the number of lines of code generated by the tool and a summary of the files. Most of the files are characterized by a constant amount of code. There are only two files (*questionnaire.html* and *controller.py*) where the amount of code grows with the number of questions. This variation is due to the fact that both implement the details of the questions. *Questionnaire.html* is responsible for displaying the questions to the end user and *controller.py* handles the requests and saves the questions on the database. The distinction between the files with variable and constant amount of code was considered to be an important factor since most of the benefits of the Domain Specific Languages stem from this variation [6]. The table shows that every additional ten questions appear to generate approximately a further six hundred lines of code. Table 2, on the other hand, presents the amount of code which was automatically generated and which is dependent on a specific Cloud Platform. This code is mainly responsible for managing Cloud based data models. It can be assumed that in order to develop applications that run on the Cloud, an additional development effort is required.

Both Table 1 and Table 2 confirm that the proposed method for developing Cloud based applications improves significantly the development process in terms of the generated code. In the case where a Domain Specific Language is applied, no manual coding is required.

B. Time of development

This section deals with the advantages of the tool in terms of the time needed to generate a fully working questionnaire application. Time was measured both for the automatic as well as the manual development of questionnaires. For the sake of convenience and expediency, the measurement for the manual development was limited to one measurement per questionnaire with a variable number of questions per manual development time.

ıme:	Five Questions	Ten Questions	Twenty Questions	Thirty Questions
ɔnnaire.html	204	478	986	1399
ler.py	183	258	414	565
eet.css	73	73	73	73
.py	59	59	59	59
y	26	26	26	26
tml	18	18	18	18
ml	11	11	11	11
·aml	11	11	11	11
l:	585	934	1598	2162

Table 1: Lines of code generated automatically

Table 3 presents time values, which are needed to develop a fully working web based questionnaires. The proposed tool appears to speed up the process of application development. It should also be noted that the estimated values for manual development given above are optimistic and are valid for experts in programming Cloud platform. It is assumed that the development process does not run into difficulty. In addition, these values do not include the time of requirements gathering. This is not an issue in automated development as the developer is usually a domain expert.

C. Maintainability

The difficulty level and time required for the introduction of changes to existing applications was also investigated. The investigation compared manual and automated ways of modifying applications. More specifically, the time required for making alterations to existing Cloud was measured, based on both the manual and the automated code generated for the cloud. The evaluation showed that for the same scenario of changes, the automated development was in some cases more than eight times faster than manual coding. Moreover, in the case of the automated development scenario, the issue of difficulty does not arise since the Cloud application developer does not need to posses any programming expertise.

No of Questions	5	10	20	30
Line of Code	290	365	521	672

Table 2: Code generated automatically for the Cloud

D. Productivity

As Kelly and Tolvanen [6] stated, it is not always beneficial to provide Domain Specific Languages. The costs of language development may exceed the costs of manual development. This is particularly valid when a Domain Specific Language is built for the production of a small number of items of one family.

Type of development:	Five Questions	Ten Questions	Twenty Questions	Thirty Questions
Manual	101 min	121 min	146 min	177 min
Automated	3 min	7 min	11 min	19 min

Table 3: Time required for the development of questionnaires

The chart in Figure 7 depicts the time, which is required to develop web-based questionnaires in a manual and in an automated manner. It presents the number of software solutions for one family, which have to be developed to achieve a return on investment from the application of the Domain Specific Language. It was estimated that fifty-six man hours (3360 minutes) were needed in order to develop a web based Domain Specific Language for development and deployment of questionnaire applications.

The chart shows that approximately for less than twenty questionnaires it would be more beneficial to adopt a manual approach. In the, case, however, where more than twenty questionnaires are generated, it might be more advantageous to develop Domain Specific Language for the automatic development. For one hundred questionnaires, the difference in terms of time of development becomes significant. The development of one hundred questionnaire applications manually takes approximately three times more time than the automatic process.

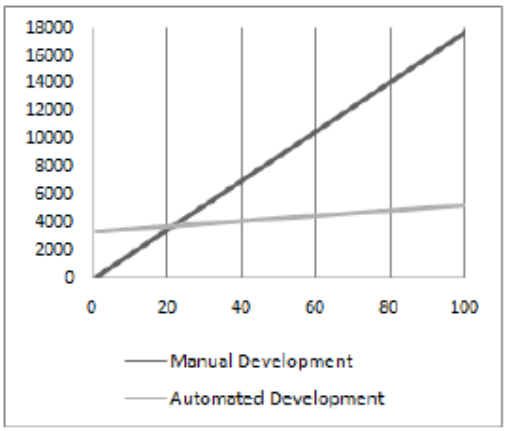

Figure 7 Manual and automatic development

E. Usability

A domain expert who is a Marketing Manager at Meji Media Ltd, and has used questionnaires as tools for market research assessed the usability of the proposed tool. According to the expert, there are two ways of building questionnaires: either by hiring a software development company or by using existing web based services. The former approach is not cost and time effective, whilst the latter does not offer much flexibility. Moreover sensitive data collected from respondents may be kept on the servers controlled by the service provider.

The proposed solution offers an innovative method of developing questionnaires. It enabled the creation of questionnaires in a time and cost effective manner. It was also considered to be more flexible than many existing web based services. Most services offer a way of creating questionnaires based on classic HTML forms, with limited scope for deciding the order of the questions or the answers, and for creating dependant questions as easily as with the proposed tool.

The other feature, which is considered to be an advantage over existing web-based services, concerns the way the questionnaire applications are deployed. Within a few seconds the application can run on an external Cloud server. Additional benefits include the enhancement of security and privacy since the data is not kept in the servers of the service owner.

VIII. CONCLUSION

A framework for developing Cloud based applications was presented and its application illustrated by a case study. The evaluation of the framework and the resulting tool has shown that this approach can be effective in addressing many of issues that hinder the wider adoption of Cloud. These include complexity, development time and cost ineffectiveness. This was achieved in two stages. First, the Domain Specific Language was implemented and deployed as a SaaS. Second, the SaaS is made accessible to designers for creating applications on the Cloud.

It was also demonstrated that the application of Domain Specific Languages enhances the process of developing and deploying applications seamlessly on Cloud. We consider that Domain Specific Languages offer a valid solution for delivering Cloud based applications in the form of Software as a Service.

REFERENCES

[1] M. Armbrust, R. G. A. Fox, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, University of California, Berkeley, Feb 2009. URL http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html, 2009.

[2] C. Boulton, "Forrester's Advice to CFOs: Embrace Cloud Computing to Cut Costs. eWeek.com, 2008.

[3] M. J. Turner and F. Gens, "Cloud Computing Drives Break through Improvements in IT Service Delivery, Speed, and Cost ftp://ftp.software.ibm.com/software/pdf/au/217961.pdf," 2009.

[4] G. V. M. Evoy and B. Schulze, "Using clouds to address grid limitations," in *6th international workshop on Middleware for grid computing*, 2008, pp. 1-6.

[5] G. Lawton, "Developing software online with platform-as-a- service technology," *Computer,* vol. 41, pp. 13-15, 2008.

[6] S. Kelly and J.-P. Tolvanen, *Domain-Specific Modeling - Enabling Full Code Generation*: Wiley-IEEE Computer Society Press, 2008.

[7] S. Cook, G. Jones, S. Kent, and A. C. Wills, *Domain-Specific Development with Visual Studio DSL Tools*: Addison-Wesley, 2007.

[8] T. Kosar, P. E. M. Lopez, P. A. Barrientos, and M. Mernik, "A preliminary study on various implementation approaches of domain-specific language," *Information and Software Technology,* vol. 50, pp. 390-405, April 2007.

[9] A. V. Deursen, P. Klint, and J. Visser, "Domain-specific languages: An annotated bibliography," *ACM SIGPLAN Notices,* vol. 35, pp. 26-36, 2000 2000.

[10] C. Weyer, "Introducing Domain Specific Languages," 2006.

[11] L. Youseff, M. Butrico, and D. D. Silva, "Toward a unified ontology of cloud computing," in *Grid Computing Environments Workshop (GCE '08)*, 2008, pp. 1-10.

[12] Google, "Google. What Is Google App Engine? URL http://code.google.com/appengine/docs/whatisgoogleappengine.html, " 2009.

[13] A. Dubey and D. Wagel, "Delivering software as a service, available at www.mckinsey.de," 2007.

[14] D. Jacobs, " Enterprise software as service 2005," *ACM Queue,* vol. 3, pp. 36-42, 2005.

[15] M. Turner, D. Budgen, and P. Brereton, "Turning Software into a Service," *Computer,* vol. 36, pp. 38-44, 2003.

[16] K. Sledziewski, "A DSL for modelling and code generation in Cloud," in *School of Computer Science, University of Birmingham, UK*, 2009.