Software Engineering Applied to Cloud Computing Environments: Improving Scalability of Cloud-based Solutions

Ivana Stupar, Goran Kopčak, Darko Huljenić
RWCE ETK Research and Innovation
Ericsson Nikola Tesla d.d.
Zagreb, Croatia
{ivana.stupar, goran.kopcak, darko.huljenic}@ericsson.com

Abstract—Cloud computing is widely known as a paradigm of virtually unlimited resources. However, this perception of cloud computing also led to the belief there should not be much effort invested into achieving scalability in cloud computing environments, other than just hosting the service in the cloud. This paper brings a short overview of the software engineering problems specific for cloud computing, as well as solutions given by the CloudScale project, i.e. proposed method and tools needed to address scalability issues in the cloud. The paper concludes with a short overview of the results of using described solutions on an industrial use case.

Keywords—cloud computing; scalability engineering; elasticity; metrics;

I. INTRODUCTION

Cloud computing is, according to the Gartner definition [1], “a style of computing in which scalable and elastic IT-enabled capabilities are delivered as a service using Internet technologies”, and is commonly perceived as a paradigm of virtually unlimited resources. This perception of cloud computing often leads to belief that cloud computing itself guarantees solution to system scalability issues. Certain mechanisms offered by cloud computing, such as dynamic allocation of computing resources, do enable scalability of the systems in the cloud. However, elasticity offered by the cloud environment cannot be fully exploited if the system scalability issues are not addressed.

Scalability issues of services in cloud computing are often not easy to detect, especially in composed services with complex structure. Additionally, performance on different layers of the cloud stack often depends on underlying layers [3], making it even more difficult to identify a point in system that is responsible for scalability problems. Even when the system is scalable in its current state, it is not an easy task to predict its behavior in future. Dynamic system usage requires the need for adaptation of system during its lifetime meaning that, at certain point, it might require architectural or deployment-related changes in order to better accommodate to its usage profile. Additional problem is to accurately predict the cost of the service, and determine how the cost of the service scales [4] considering heterogeneous cloud economic environment with many different pricing models offered by the cloud service providers.

II. TOOLS AND METHOD FOR ADDRESSING SCALABILITY ISSUES IN THE CLOUD

Issues mentioned in the previous section are addressed by EU FP7 project CloudScale [2]. Project goal is to enable scalability engineering throughout the whole lifecycle of the system. Tools and method are provided in order to identify and address scalability issues, and enable the exploitation of cloud elasticity.

A. Metrics

CloudScale tools use a set of metrics defined based on externally observable properties of the system with the focus on scalability, elasticity, and efficiency.

Scalability Range metric defines a range of workload, and Scalability Speed specifies the maximal change rate of the workload, that the system can handle while achieving its Service Level Objectives (SLOs). Mean Time to Quality Repair specifies the mean time it takes for a system to be able to achieve its SLOs again when the workload changes. Number of SLO Violations measures the number of times system's SLOs are violated as a result of the workload changing at the specified change rate. Resource Provision Efficiency is measuring the ratio of actual resource utilization to resource demand during the workload change in order to determine system's resource efficiency. Another metric related to efficiency is Marginal Cost that specifies cost of serving one additional workload unit.

B. CloudScale Tools

CloudScale offers several tools integrated in the CloudScale environment, serving as a base for CloudScale method described in the following subsection.

Extractor generates component model of the service based on its source code provided as the Extractor tool input. The generated model can be manually enhanced before it serves as an input for the Analyser tool.

Analyser uses the system component model as an input and detects scalability issues and bottlenecks, using the metrics defined in previous subsection. Analyser can be used to
Spotter is a modeling tool that will be used to identify scalable system requirements. If scalability issues are detected, the system is ready to be realized and deployed in a cloud environment.

The CloudScale method (Figure 1) aims at providing scalability engineering support at design-level of the system and through the system evolution. The input to the CloudScale method is the system scalability requirements specification created as a result of requirements identification. The next step is providing the component model of the system either through reverse engineering or by creating the model manually. If the source code of the system exists, the Extractor tool can automatically provide the system component model. Users have the possibility to enhance the generated model manually. The created model of the system is further being analyzed by the Analysers tool. Analysers predicts the scalability of the system and compares the predicted values to the requirements provided as the input of the method, and checks if the scalability requirements are met. If no scalability issues are detected, the system is ready to be realized and deployed in the cloud environment.

In case Analysers has detected scalability issues in the modelled system, the Spotter tool will be used to identify the component of the system causing the issue, i.e. the component that is being the scalability bottleneck of the system, based on the source code and the model of the system. Problems detected by Spotter can be solved using the anti-pattern solutions or by redesigning the system. After applying the solutions, the system will again be analyzed with the Analysers tool, making sure the scalability issues were resolved.

After the system has been realized and deployed, it can enter the operation and monitoring phase where measurements of the scalability-related parameters are collected. Dynamic Spotter is able to detect anomalies and find scalability issues by monitoring the system (“Spotting By Measuring”) and informing software engineers about the possible problems and state of the running system. Such approach of continuous monitoring is important for scalability evolution of the system once it becomes operational.

III. USE CASE AND CONCLUSION

The industrial use case used for CloudScale solutions validation provided by Ericsson Nikola Tesla is Electronic Health Record (EHR), a part of information exchange system supporting healthcare delivery processes deployed on a national level in Croatia. EHR stores medical information and shares the data with healthcare providers, patients and government institutions. Currently EHR is not hosted in the cloud and scalability is assured for limited system size, by mechanisms such as database clustering and replication. Migrating EHR to the cloud would reduce operational costs and ensure scalability with variable workloads. CloudScale solutions will be used to evaluate system scalability, provision optimal amount of cloud resources, and monitor system properties during the operational phase. One of the most scalability-critical parts of EHR is the database because the system is data-intensive, but issues can also occur in the load balancing, caching, and resource pooling.

The main benefit of the CloudScale project for the EHR is scalability engineering support, especially in cases of unexpected increased system workload (e.g. large number of patients during flu season). The EHR service model was generated using the Extractor tool, and scalability issues resulted from inappropriate design and architectural patterns are identified using the CloudScale tools. Additional expected benefit that is going to be tested in the next phase is an easier process of migrating the EHR service to the cloud using the CloudScale method providing the best practice solution and scalability know-hows.

ACKNOWLEDGMENT

This work was supported by the European Union Seventh Framework Programme (FP7/2007-2013) under grant no 317704 (CloudScale).

REFERENCES

Abstract — Social networks and their potential change our life. In 2014, 74% of Internet users use social networking sites. Although social networking started in early 2000s as a service primarily oriented towards Generation Y, more than 71% of online adults started to use Facebook by 2014. In this paper we will pinpoint major differences between traditional application development and social application development from the perspective of software engineering practice.

I. EXTENDED ABSTRACT

Social networks and their potential change our life. In 2014, 74% of Internet users use social networking sites. Although social networking started in early 2000s as a service primarily oriented towards Generation Y, more than 71% of online adults started to use Facebook by 2014. Furthermore, more than 40% of cell phone owners use a social networking site on their phones [1]. Finally, US Internet users spend three times more minutes on blogs and social networks than on using e-mail service [2]. How to cope with this new paradigm of connected people and enable fast development of new applications based on social networks [3, 4, 5]?

In this paper we will pinpoint major differences between traditional application development and social application development from the perspective of software engineering practice. In Figure 1 a basic lifecycle of the traditional application development is presented through four steps: i) requirement specification; ii) application design; iii) application coding; and iv) application evaluation. Furthermore, eight major software engineering challenges introduced in the lifecycle with development of applications based on social networks are identified. These challenges will be elaborated in more details in the remainder of this paper.

1) Authentication and authorization (Facebook)

Facebook's authentication flows are based on the OAuth 2.0 protocol. OAuth is an open protocol to allow secure application authorization through a simple and standard method. The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to a HTTP (Hypertext Transfer Protocol) service.

When someone connects with an app using the Facebook Login, the app will be able to obtain an access token which provides temporary secure access to Facebook APIs (Application Programming Interface). The access token is an opaque string that identifies the user, app or page and can be used by the app to make graph API calls. Because of privacy checks, the majority of API calls on Facebook need to include an access token. There are different types of access tokens to support different use cases: i) User Access Token; ii) App Access Token; iii) Page Access Token; and iv) Client Token.

2) API rate limit

Facebook's information on limitation of the frequency certain app uses it API is really vague and following information can be found in different sources:
- 600 calls per 600 seconds per access token per IP (Internet Protocol) address;
- 100M calls per day per app;
- 10K calls per user token per day.

Twitter rate limits in the API version 1.1 are divided into 15 minute intervals, which is a change from the 60 minute blocks in version 1.0. Additionally, all v1.1 endpoints require authentication, so no longer will there be a concept of unauthenticated calls and rate limits. While in API v1.0 an OAuth-enabled application could initiate 350 GET-based
requests per hour per access token, API v1.1’s rate limiting model allows for a wider range of requests through per-method request limits. There are two initial buckets available for GET requests: 15 calls every 15 minutes and 180 calls every 15 minutes.

Instagram global rate limits are applied inclusive of all API calls made by an app over the 1-hour sliding window, regardless of the particular endpoint, and that is 5000 requests per hour per access_token or client_id overall.

3) Choosing suitable programming language

The answer to the question how to pick the right programming language is based on social network for which we want to build apps. The best practice is to consider which official SDKs (Software Development Kits) are available and choose one or more.

4) Tackling frequent API changes

Despite its popularity, Facebook API changes are radical and infrequent. In Figure 2 it is shown how those changes came to effect in terms of versions used.

![Figure 2. Changes to the Facebook API](image)

5) Developer and application registration

Developer account is required for building social network apps and every app need be registered on a social network developer page. For example, a Facebook developer account extends developer Facebook personal account with access to Facebook developer tools including Facebook application management.

6) Network effect in the application evaluation

In order to create what is known as the network effect [6], i.e., to obtain a critical mass of users of a certain service, one needs to consider limitations and obstacles in doing so. In the case of social app evaluation, the network effect is a needed phenomenon in order to have a valid evaluation. Having a small number of users (e.g., less than 100) limits the variations and possibilities in the input sample. On the other hand, having a larger input sample (e.g., more than 500) enables variety in how numerous participants’ connections are, how interconnected they are and how social networking graph is detailed in terms of its centrality, span, and other metrics important for analysis.

7) Privacy concerns

One would probably seriously question the problem of privacy in social networking services [4] with 83 million accounts that are alleged spammers on Facebook [5]. On the other hand, fewer than 30% of large organizations will be blocking social networking services by the end of 2014 [6]. When compared to 50% in 2010, there is a clear trend in this area, probably due to increased control over published user information (e.g., Facebook’s Activity Log) and the chance that users got accustomed to these features, which Facebook fully enables only until recently [7]. When it comes to privacy, it is important to be transparent about exactly what information users share, with whom and when they share it.

REFERENCES

xtUML in Action: Advantages and Challenges

Robert Inkret  
Research department, R&D  
Ericsson Nikola Tesla  
Zagreb, Croatia  
robert.inkret@ericsson.com

Marijan Zemljic  
Research department, R&D  
Ericsson Nikola Tesla  
Zagreb, Croatia  
marijan.zemljic@ericsson.com

Abstract— Inherent complexity and requirements of the telecom domain make development of telecom applications and systems extremely complex and time consuming process. There is a clear need to make development process more efficient by moving developer focus back to the application logic and by shortening debug/test feedback cycles. The presentation will give overview of experience in development of components for xtUML and results of application in the commercial project. For long time companies are waiting for tool that will enable smart software production but we still have some challenges.

I. EXTENDED ABSTRACT

Inherent complexity and requirements of the telecom domain make development of telecom applications and systems extremely complex and time consuming process. Complexity of today's telecom platforms had long ago reached a level where application developers and engineers spend substantial amount of time handling platform complexity instead on focusing on the application logic, which actually brings added value. Moreover, sheer size of the systems makes build and debug process very lengthy. Hence, developers usually must wait a non-negligible amount of time to build, test and get feedback on their work. There is a clear need to make development process more efficient by moving developer focus back to the application logic and by shortening debug/test feedback cycles.

Modeling is a well-known and proven technique for handling the complexity by raising the level of abstraction. E(x)ecutable and (t)ranslatable UML (xtUML) is a graphical software modeling language which has a great potential in dealing with above mentioned challenges. By combining a well-defined subset of UML with precise executable action semantics and timing rules, xtUML makes UML diagrams unambiguous so they can be both precisely executed and translated. xtUML facilitates development on a higher abstraction level and model execution as a mean of gaining early feedback on the design/architecture choices. By separating domain and implementation concerns it facilitates more efficient development process as application developers can focus on domain problems rather than on implementation details. All necessary implementation level details such as design rules, patterns and target technologies are incorporated into model compilers developed by platform specialists.

xtUML process comprises of four steps which could be repeated iteratively in an incremental manner until all identified requirements are satisfied. In the first step application (or system) is partitioned into a self-contained domains based on the subject matter boundaries. This implies definition of components and interfaces through which those components communicate with its environment or other components. In the second step each domain (component) is further elaborated using three different views/diagrams. The class diagram is used to define the domain abstractions (data), relationships between them and applicable constraints. Discovered abstractions represent the real world concepts that could be found in the domain. Class modeling is considered the most important step in the xtUML process as good class diagrams expose great deal of information on the domain (not only static but also dynamic) and lead to a clear and precise state diagrams and action code later on. State diagrams are used to capture the lifecycles of discovered concepts and finally action code is written in class operations, transitions and states in order to define processing. In the third step application models are executed either against manually created or automatically generated (from requirements) modeled test cases to verify the functionality. Finally, when application models fulfill given requirements, models are translated compilation ready code by means of a model compiler, also known as model translator.

In this presentation we bring some of our experiences in applying the model driven xtUML technology onto the development of a real telecom product, an IMS logical node implementing SIP (Session Initiation Protocol) Back to Back User Agent (B2BUA) functionality. Functionality of the case study application includes anchoring of the communication sessions to facilitate session continuity when a user, for example, changes the access network.

We explain some of the challenges we faced and argue that probably the biggest and most important challenge in transforming the traditional development process towards the model driven one is the required paradigm shift. We also cover some of more tangible issues such as model integration with existing legacy systems, or problems related to model compiler support required for specific platforms, like serialization and deserialization of a model, and upgrade of already deployed applications. Finally, at the end, we present the list of achievement and benefits one can expect when applying the model driven xtUML technology onto product development.
Towards semi-automated software development
The S-CASE approach

Abstract—Development tasks least supported by development tools are analysis of user requirements to form a conceptual model of the envisioned system, and the search for reusable software artefacts that match those requirements. This paper introduces efforts towards creating a development environment capable of extracting system specifications architecture from user requirements, to be used for workflow synthesis, composition of reusable software artefacts (web services) and generation of prototype source code.

Keywords—automated software engineering; software requirements; multi-modal processing; development tools

I. INTRODUCTION

Software engineering, as practiced today, is forced to deal with increasing complexity of user requirements and software development and the customers’ desire to reduce the time to market. Emerging technologies promise simplified and automated development, but the necessity to rapidly turn a conceptual idea to a running prototype is beyond any current technology. Automation of software engineering has focused on producing quality software in the shortest time, with developer tools accelerating most common development tasks, from conceptualization and development towards deployment and maintenance.

The tools support in the earliest stages of design and development is limited: the translation of requirements to system design and implementation is still a manual task. Automated translation of requirements to a fully implemented prototype or even prototype skeleton ready to be completed by the developers is yet to be achieved. An important step towards this goal is a large variety of existing software artefacts available in open source repositories, libraries or deployed web services.

Searching for available software artefacts with a level of confidence required to include them in developed software is hindered by the level of experience and the extensiveness of testing to ascertain the quality of the artefact. The ability to establish a relation from the original requirements for the artefact and the requirements for the software using the artefact would greatly reduce the time spent evaluating third-party software, but also widen the scope of the search for artefacts from well-known and proven solutions to include lesser-known solutions and recent additions.

II. INTRODUCING S-CASE

The vision of S-CASE is to provide technologies and tools for developers that will support the insertion of rough system requirements in a variety of structured or semi-structured formats, such as plain text in a natural language, software models (UML models), or hand-drawn images of software models. After the multi-modal requirements are processed, the tool-chain will enable seamless generation of draft software prototypes based on reuse of existing artefacts to establish the foundation for complete software development.

S-CASE is a European Framework 7 project[1] that is expected to have a significant impact on the reduction of the time between the conceptualization of a software system and its first prototype, with a final goal of reducing development costs. The project is seen as a rapid prototyping realm aiming at automating the tasks of extracting system specifications and low-level architecture, and discovery and synthesis of composite workflows of software artefacts from distributed open source and proprietary resources matching the system requirements in the best possible way. Resulting methods and tools will give the developers the power to see their ideas quickly transformed into a prototype that can be evolved to refine the envisioned concept.

III. DEVELOPMENT PROCESS

The influence of S-CASE on the software development process is shown in Fig. 1. A generic software development process is shown on the left-hand side and the modification of the same process when using S-CASE tools is shown on the right-hand side. For the purpose of this argument, an iterative-like process is used as an example to note that after a prototype

---

This project has received funding from the European Union’s Seventh Framework Programme for research, technological development and demonstration under grant agreement no 610717.
Fig. 2. Proposed user interface for the developer

requirements phase. S-CASE approach is not constrained to any particular development method; instead it aims to replace manual tasks with automated or semi-automated tools.

In an envisioned development workflow, the developer would follow a simple workflow: upload multi-modal requirements for analysis, review the results and obtain a workflow proposal, specify preferences and constraints to obtain proposals of artefact composition, select the best proposal for code generation, obtain and revise application models and generate source code. The prototype source code can serve as a skeleton for further development, or as a paper prototype to verify the original idea. It can be noted that the latter stages of the workflow rely on model-driven engineering approach that derives the final source code in a series of model transformations starting from more generic, architecture- and platform-agnostic models, towards architecture- and platform-specific models.

The key steps in the workflow, and the ones that have most influence on the perceived quality of the result, are the generation of the workflow proposal and the artefact composition proposals. Inputs to workflow generation are the analyzed multi-modal requirements. The resulting workflow is used as an input to the artefact composition phase, which will query a repository of semantically annotated artefacts (web services) to find the best match to the requirements, or compose a new service if necessary. The developer may be asked to assist in the search.

IV. DEVELOPER TOOLS

A sketch of a proposed integrated development environment shown in Fig. 2. uses the following views:

1. Navigation view – standard navigator showing artefacts, e.g. requirements, models, workflows, service mappings, notes (or other metadata like links to other repositories), and generated code.

2. Dashboard – outlines the development workflow discussed earlier in a sequence of steps to ensure consistency.

3. Search – S-CASE-enabled search tool towards the artefact repository, with the ability search for queries posed as questions in a natural language.

4. Central area – used for editing; shown here is the recognition of a hand-drawn UML activity diagram.

The layout of the user interface is based on the de-facto standard layout of popular development environments.

V. CONCLUSION

In this paper we briefly introduced the concept of S-CASE and its goal of achieving a level of automation in production of software prototypes reusing previously available software services. In the discussion we focused on the developers’ perspective and omitted supporting elements of the envisioned realm – repositories, service annotation tools, and the underlying methods for multi-modal processing, semantic annotation and matchmaking. The proposed development environment described here is preliminary and will be revised to better suit the user expectations.

REFERENCES