Design and Implementation of a Social Networking Architecture for Cloud Deployment Specialists

Christos Papoulas

Thesis submitted in partial fulfillment of the requirements for the

Masters' of Science degree in Computer Science

University of Crete
School of Sciences and Engineering
Computer Science Department
Voutes University Campus, P.O. Box 2208, Heraklion, GR-70013, Greece

Thesis Advisor: Prof. *Manolis G.H. Katevenis*

This work has been performed at the University of Crete, School of Science and Engineering, Computer Science Department and at the Institute of Computer Science (ICS) - Foundation for Research and Technology - Hellas (FORTH), Heraklion, Crete, GREECE.
The work is partially supported by the PaaSage (FP7-317715) EU project.

University of Crete
Computer Science Department

Design and Implementation of a Social Networking Architecture for Cloud Deployment Specialists

Thesis submitted by
Christos Papoulas
in partial fulfillment of the requirements for the
Masters' of Science degree in Computer Science

THESIS APPROVAL

Author: ______________________________
Christos Papoulas

Committee approvals: ______________________________
Manolis G.H. Katevenis
Professor, Thesis Advisor

____
Kostas Magoutis
Assistant Professor, University of Ioannina, Thesis Co-advisor

____
Dimitris Plexousakis
Professor, Committee Member

Departmental approval: ______________________________
Antonis Argyros
Professor, Director of Graduate Studies

Heraklion, October 2015

Abstract

A new discipline at the intersection of the development and operation of software systems known as DevOps has seen significant growth recently. A class of DevOps engineers that are experts in the configuration and deployment of applications on cloud environments (also known as cloud deployment specialists), increasingly use automated deployment and release-engineering tools like Chef and IBM Bluemix to configure and deploy their applications. Despite the advent of automated mechanisms, reasoning about good deployments still requires interaction with experts, often through discussions on online technical forums and social networks.

Within the DevOps community, communication on application structure and cloud deployment tradeoffs could become more effective by bridging social networking technologies with knowledge present in global community-sourced information repositories. In this work we propose a social networking architecture (named after the PaaSage EU project) where cloud deployment specialists can express applications and their requirements as software models (using the Cloud Application Modeling and Execution Language or CAMEL), capture execution results from various multi-cloud platforms into a specifically-created information repository, and communicate with their peers on design and deployment issues, including deployments tradeoffs.

The implementation of the PaaSage social networking architecture provides users with information mined from collected executions of distributed applications, facilitating their choice of deployment platform based on various criteria (such as cost effectiveness). Our investigation explores and evaluates several techniques to improve the scalability of the platform. Finally, to better direct cloud deployment specialists to possible answers to their questions we leverage topic-classification tools to associate user questions with related questions-and-answers (some of which may contain the results of queries on the historical execution information). Our implementation is in pilot use within the PaaSage project since March 2015.

Περίληψη

Μια νέα τάση που γεφυρώνει την ανάπτυξη με την λειτουργία των κατανεμημένων εφαρμογών, γνωστή ως DevOps, έχει γνωρίσει σημαντική ανάπτυξη τα τελευταία χρόνια. Μηχανικοί DevOps με εξειδίκευση στην προσαρμογή και εγκατάσταση εφαρμογών σε περιβάλλοντα υπολογιστικού νέφους (γνωστοί και ως cloud deployment specialists), χρησιμοποιούν όλο και περισσότερο συγκεκριμένα εργαλεία για την εγκατάσταση και λειτουργία των εφαρμογών τους, όπως το Chef και το IBM Bluemix. Παρά την σημαντική αυτοματοποίηση που προσφέρουν αυτά τα εργαλεία, η εύρεση των χαρακτηριστικών μιας επιτυχούς εγκατάστασης κατά περίπτωση απαιτεί συζήτηση με ειδικούς, συχνά σε διαδικτυακά τεχνικά φόρουμ και κοινωνικά δίκτυα.

Εντός της κοινότητας των μηχανικών DevOps, η συζήτηση γύρω από την δομή των εφαρμογών και την επίδραση των διαφόρων παραμέτρων εγκατάστασης τους σε υπολογιστικά νέφη μπορεί να γίνει πιο εποικοδομητική αν οι τεχνολογίες κοινωνικής δικτύωσης εμπλουτιστούν με την γνώση που υπάρχει σε αποθετήρια δεδομένων χρηστών της κοινότητας DevOps (όπως π.χ. το Chef Supermarket). Σε αυτήν την εργασία προτείνουμε μια αρχιτεκτονική κοινωνικής δικτύωσης (που παίρνει το όνομά της από το PaaSage EU project) όπου οι χρήστες μπορούν να περιγράψουν τις εφαρμογές και τις απαιτήσεις τους ως μοντέλα εφαρμογών (χρησιμοποιώντας την Cloud Application Modeling and Execution Language ή CAMEL). Η πλατφόρμα υποστηρίζει την συλλογή αποτελεσμάτων εκτελέσεων των εφαρμογών σε πολλαπλά υπολογιστικά νέφη και την αποθήκευσή τους σε ειδικά σχεδιασμένο αποθετήριο δεδομένων.

Η υλοποίηση της αρχιτεκτονικής κοινωνικής δικτύωσης PaaSage παρέχει στους χρήστες πληροφορίες που έχουν προέλθει από ιστορικά δεδομένα εκτελέσεων κατανεμημένων εφαρμογών, διευκολύνοντάς τους στην επιλογή εγκατάστασης (ποιές υποδομές νέφους, ποιοί τύποι εικονικών μηχανών, κλπ) με βάση σύνθετα κριτήρια, όπως η ανάλυση κόστους-αποτελεσματικότητας (cost effectiveness). Στην εργασία διερευνούνται και αξιολογούνται τεχνικές για την βελτίωση της κλιμακωσιμότητας της πλατφόρμας. Τέλος, για την καλύτερη καθοδήγηση των χρηστών που θέτουν τεχνικές ερωτήσεις, στις βέλτιστες πιθανές απαντήσεις, αξιοποιούμε συστήματα κατηγοριοποίησης θεμάτων για την συσχέτιση ερωτήσεων των χρηστών με αποθηκευμένες ερωτήσεις και απαντήσεις (μερικές από τις οποίες περιλαμβάνουν αποτελέσματα από επερωτήσεις (queries) επί ιστορικών δεδομένων αποτελεσμάτων εκτελέσεων εφαρμογών). Η υλοποίηση είναι σε πιλοτική λειτουργία εντός του έργου PaaSage από τον Μάρτιο του 2015.

Acknowledgements

There are so many people that I would like to thank, each one helped me in their own special way.

First of all, I would like to thank my co-advisor Professor Kostas Magoutis for guiding and having trust in this work, but mainly I would like to thank him for his continuous enthusiasm and willingness to help me at every stage of this thesis. I would also like to thank my advisor Professor Manolis Katevenis.

I want to acknowledge Antonis Papaioannou, Emmanouil Papoutsakis, members of PaaSage team for their valuable feedback on my work as well as Maria Korozi, Asterios Leonidis and Stavroula Ntoa for the collaboration during the jisa journal publication and the design of the user interface.

I need to express my gratitude to the University of Crete and the Department of Computer Science; as well as the Institute of Computer Science of the Foundation for Research and Technology for supporting me.

During my time in Heraklion, I have been fortunate to enjoy the friendship of a number of special people. I would like to give my appreciation to my friends, Theodoros Sfakianakis, John Bikouvarakis, Christos Mousamatas and Apostolis Matthaiakis for the encouragement and the support during my undergraduate and graduate studies as well as for the great moments we had together during these years.

I would also to thank my partner, Natalia Athanasiadi, for all her love, support, patience and encouragement. It was her continued encouragement that got me through this process.

This thesis is dedicated to the memory of my mother Chariklia Laggourani who passed away in 2015 at the age of 63. She passed away just two months before this thesis was finished and at the last of her days she was proud of my achievements in completing my master's thesis.

Christos Papoulas

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A new discipline at the intersection of the development and operation of distributed software systems, known as DevOps [1], has seen significant growth recently. The DevOps community comprises engineers from the development and operations fields, including programmers, testers, quality assurance staff [2], system administrators, and others. These professionals require different tools for carrying out their daily tasks, such as building applications, testing, deploying routines, configuration, automation of utilities, tracking and versioning in systems. With the advent of cloud computing, a new breed of DevOps professionals, cloud deployment specialists has emerged. Cloud deployment specialists are responsible for the configuration, provisioning, deployment, testing, and lifecycle management of hosted application solutions. They typically have thorough industry knowledge on all areas related to the secured hosting and lifecycle management of cloud applications.

A number of automated tools are available for cloud deployment specialists today. One such tool is Chef and its associated repository, Chef Supermarket [3]. Chef is an application configuration framework that makes it easy to deploy applications on servers on any physical, virtual, or cloud location. The representation of the application resembles code using Chef cookbooks. Another such tool is IBM Bluemix [4], a development and support platform for communities of DevOps users wishing to compose distributed applications out of components drawn from libraries and deploy them at IBM-provided and supported cloud infrastructure. Both tools however provide only minimal community interaction facilities and do not collect nor leverage any past (historical) experience on application deployments.

Today DevOps cloud deployment specialists exhibit the need to communicate through traditional social networks and other online fora, to exchange views and potential solutions on the cloud deployment of applications. The current status in this space, however, can be improved. A motivation for this thesis was the need to improve the efficiency of the interaction and communication between cloud deployment specialists, coupling human interaction with knowledge present in crowdsourced repositories of information, as that would provide them with expert advice and input that is not currently available through the use of automated tooling.

The topic of this thesis is thus the design and implementation of a social networking architecture targeted for cloud deployment specialists engineers. This social network architecture binds several social networking concepts such as personal messaging, groups, live feeds, etc. with software-engineering concepts such as the composition, deployment, monitoring, and analysis of executions of applications. The architecture integrates social networking with knowledge present in a repository of cloud applications and infrastructure description based on Cloud Application Modelling and Execution Language (CAMEL) [5]. Among the range of possible DevOps tasks, the design of the social network architecture puts special focus on one of the most challenging tasks of cloud deployment specialists: selecting the most appropriate deployment configuration for an application. This is especially challenging in a multi-cloud setting due to the large diversity of deployment possibilities and tradeoffs.

Currently DevOps users work with a small set of well-understood deployment options, missing on opportunities for improving performance, reliability and/or lower cost. Investigation of new options involves time consuming testing over new infrastructures. Discussing those topics with the community in online social or technical forums may provide insight over deployment options; however, the answer to a hard question often needs to be backed by experimental data that is not readily available.

An integrated environment such as proposed in this thesis, comes to solve the above issues by enriching user interactions with structured references to applications and their components, execution data, and mined knowledge from real deployments. Mined knowledge can be combined with user activity and profiles to provide personalized suggestions and hints. An improved mode of user interaction is expected to result in stronger incentives for DevOps users to contribute information to the underlying repositories. Richer content should lead to better quality of mined knowledge, benefiting the DevOps community and providing further incentive for contributions. The social networking platform is designed to be closely integrated with a set of information repositories satisfying the following requirements: (R1) handle entire applications rather than just software components; (R2) abstract application structure through software modeling; (R3) capture and analyze application runtime performance. Existing repositories used in this thesis include Chef Supermarket and the PaaSage CAMEL repository.

Broadening the focus from individual software components to entire applications and analyzing their execution data, can provide answers to many interesting questions and support community discussions and arguments with hard data. We believe that these requirements can provide software developers with strong urge to contribute, leading to the sustainability and growth of information and derived knowledge in the repository.

This thesis is structured as follows: Section 1.1 provides background on application models and the CAMEL repository developed in the context of PaaSage EU project. Chapter 2 describes previous work on the scalability of social networking platforms through caching mechanisms and on topic classification. In

Chapter 3, we describe the implementation of the social networking architecture[1], and in Chapter 4 we present our evaluation. Finally, in Chapter 5 we present our conclusions.

1.1 Background

This section presents an overview of the descriptions of the application models that are available on the social network platform and a summary of the technologies that are used by the PaaSage in order to assist the cloud deployment specialists to deploy the aforementioned application models.

Application models inside social network platform are described in CAMEL. CAMEL integrates various domain-specific languages (DSLs). DSLs provide a notation tailored towards an application domain and are based on the relevant concepts and features of that domain. As such, a DSL is a means to describe and generate members of a family of programs in the domain. These DSLs cover a wealth of aspects of specification and execution of multi-cloud applications like CloudML, Scalability rules, WS Agreement, Saloon and Historical Execution Data.

CloudML [7] is a recent approach that focuses on the provisioning and deployment of multi-cloud applications, is built upon MDE techniques and methods, and provides a models@run-time [8] environment for enacting the provisioning. WS Agreement [9] is a Web Services protocol for establishing agreement between two parties, such as between a service provider and customer. Saloon [10] is an approach that uses models to represent clouds variability, as well as ontologies to describe the heterogeneous aspect of the cloud ecosystem. The CAMEL model assembles all those DSLs as shown in Figure 1.1.

CAMEL is using the Eclipse Modelling Framework (EMF) [11] on top of the Connected Data Objects (CDO) [12] [13]. Application Models are persisted on the CDO repository. EMF is used as a building tool for the application models. CAMEL model specification is written in XMI format (XML Metadata Interchange) which is a standard for exchanging metadata information via Extensible Markup Language (XML). Thus, XMI consists the main describing system of CAMEL and EMF provides tools that enable viewing and command-based or tree-based editing of the application models.

A representation of a model using the EMF editor is shown in Figure 1.2. The cloud deployment specialist can add all the information needed for the deployment of the application. For example, the figure 1.2 shows the “Deployment Model”, which contain the configuration of CloudML. This information is needed for the application to be deployed. The “Provider Model” and the “Location Model” contains information about the cloud providers that will be used for this specific deployment of the application. This application has not any historical executions yet, but after the deployment it will be populated with those data.

[1] The user interface (UI) design and usability evaluation was the result of a collaboration with the Human-Computer Interaction (HCI) laboratory of ICS-FORTH [6].

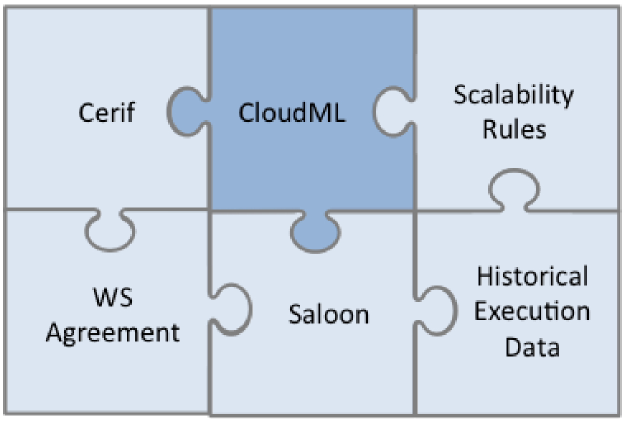
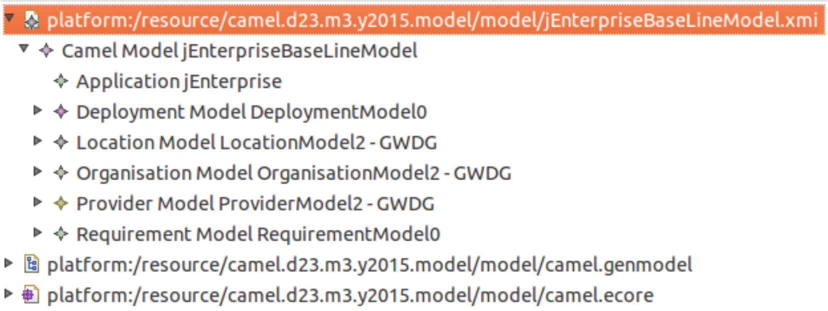


Figure 1.1: CAMEL DSLs

The CAMEL repository is currently being populated with a wealth of information from multi-cloud deployments of various distributed applications [14]. Also, SNP performs analytics over the CAMEL repository to extract knowledge about deployments characteristics that work best for certain applications and use it in the context of the professional network.



```
▼ platform:/resource/camel.d23.m3.y2015.model/model/jEnterpriseBaseLineModel.xmi
  ▼ Camel Model jEnterpriseBaseLineModel
      Application jEnterprise
    ▶ Deployment Model DeploymentModel0
    ▶ Location Model LocationModel2 - GWDG
    ▶ Organisation Model OrganisationModel2 - GWDG
    ▶ Provider Model ProviderModel2 - GWDG
    ▶ Requirement Model RequirementModel0
▶ platform:/resource/camel.d23.m3.y2015.model/model/camel.genmodel
▶ platform:/resource/camel.d23.m3.y2015.model/model/camel.ecore
```

Figure 1.2: An application model view from EMF editor

The social network platform proposed in this thesis is part of the PaaSage EU Project [15]. The PaaSage perspective is to be a tool for a cloud deployment Specialist to leverage the complex task of deploying an application to the clouds. Usually, a cloud deployment specialist could easily learn the interface and features of one Cloud provider, but it would be very costly and time consuming to leverage the development to many providers. It is a real challenge to orchestrate the simultaneous deployment to many different Clouds at the same time. The main objective of PaaSage is to assist the developer to deal with difficult deployment scenarios through automatic cloud deployment. In order to satisfy this, several components are included to the PaaSage ecosystem. The Profiler components read the CAMEL models and convert them into a constraint programming model by defining the variables of the model, their domains, and the constraints that must be satisfied by the deployment. Also, the Profiler checks all constraints of the CAMEL model and sets the domains of the variables accordingly. The reasoning component analyses the model and finds how deployment candidates should be evaluated. Once a solution has been found, the reasoning component converts the model to Cloud Provider Specific Models (CPSM) for the providers involved in the proposed deployment. The adapter component takes the CPSMs, produces and validates a configuration plan, and sends this plan to the execution ware. The execution ware [16] receives the deployment plan from the adapter and enacts the deployment of the application on the selected providers. Furthermore, the execution ware interacts with the Cloud providers, acquires the virtual machines, configures them and launches the user application on the set of those machines. Once the machines are running, the execution ware collects sensor data for the running application, triggering re-configurations if necessary.

The social network platform brings to the DevOps users a friendly interface to browse, discover, view and discuss Application Models. Furthermore, it presents a way to deploy and run these Application Models, by using the previously mentioned components under the hood, and mines their execution history data.

The key contributions of this social network platform (SNP) are the following:

- The SNP binds all the Social Networking aspects such as friends, new feeds, personal messages etc. with the engineering aspects of creating and deploying application models.

- The SNP brings the execution histories of the CAMEL applications in the light, providing the end users the ability to browse, discuss, point and find essential information needed for other applications.

- The SNP uses the best known practices both for the front end viewing system and the back end technology.

- The SNP runs on a horizontal scale architecture with memcached at the back end to reach near real time interaction.

Chapter 2

Related Work

In this chapter the related work of other professional networks is presented in Section 2.1 and their caching architecture is described in Section 2.2. In addition, the Section 2.3 describes related tools and systems that the DevOps use in order to make their deployment faster. Finally, an overview and related work of topic classification is presented in Section 2.4.

2.1 Professional and social networks

This section reviews the related work on other professional social networking platforms and what they provide. Having the requirements of DevOps users as a priority, we compare those platforms with our proposed SNP, pointing their deficiencies and filling in the gaps with our innovation.

Table 2.1 summarizes and categorizes the characteristics of the most important related approaches along the following dimensions (depicted as columns of Table 2.1):

- Which of the following key social features are supported by the platform: follow users; news feeds; groups; Q&A; personal messages;

- Does the platform rely on one or more repositories to store the following type of information: software code, software models, configuration information, execution histories; and whether these repositories are community sourced;

- Does the social networking platform leverage the repositories to provide users with specific suggestions and hints;

- Does it support application deployment?

Information Technology (IT) [17] professionals use a variety of online sources as aids in their daily tasks. Developers typically prefer community-moderated forums over vendor-moderated sites [18]. Social networks focusing on software technology in particular provide developers with the opportunity to leverage the knowledge and expertise of their peers.

One of the most popular such platforms is GitHub [19], a collaborative revision control platform for developers launched in April 2008, and arguably the largest code-hosting site in the world. GitHub provides social networking functionality such as feeds, followers, wikis and a social network graph that captures how developers work on versions of their repositories, which version is newest, etc. Gitter [20] is a related service that facilitates discussions between members of GitHub communities by providing a long-term chat integrated with code and issues. Sourceforge [21] was the first code-hosting platform offered to open-source projects. It was launched in 1999 and offered IT professionals the ability to develop, download, review, and publish open-source software. Sourceforge is similar to GitHub in its support for social features. Other similar code-hosting platforms are Google Code [22] and Microsoft CodePlex [23]. None of those platforms collect, analyze, or use information from executions of application deployments to improve the level of technical discussion between users or abstract code structure through modelling or enhance user interactions through the use of analytics over application execution histories.

StackOverflow [24] advances on earlier community-driven Q&A sites in which users ask and answer questions. Users can vote up or down questions and answers and earn *reputation points* and *badges* in return for their active participation. Although StackOverflow and GitHub address different aspects of software development (StackOverflow is not a code-hosting platform) there is a synergy and correlation between the two [25]. The proposed social network platform extends StackOverflow through the use of social networking features that enable users interested in reasoning about application deployments to use and share knowledge drawn from analyses of information repositories.

IBM's BlueMix [4] is a key component of IBM's DevOps best practices [26] for achieving rapid prototyping, automated deployment, and continuous testing of software. BlueMix encourages its users to ask their questions to StackOverflow but also includes a community forum [4] with rating of answers contributes to eventually building a basic knowledge base, similar to traditional approaches such as StackOverflow. The proposed social network platform system differs from BlueMix in its support for expressing applications as models (CloudML, CAMEL) and its use of two information repositories, the PaaSage repository of models and execution histories and Chef Supermarket, and the use of analytics over past executions to enable users to reason about application deployments. A common feature between the proposed social network platform and BlueMix is support for deployment of distributed applications.

LinkedIn is widely adopted across a range of professional communities due to its robust set of social features (and to some extent due to its use of extensive analytics over collected information [27]), LinkedIn provides no specific support for software engineering activities and thus more closely resembles traditional social networking platforms such as Facebook.

	User Interaction				Repository						
	Social features [a]	Groups	Q & A	Personal messaging	Software code	Software models	Software config	Execution histories	Crowd sourced	Repo assisted hints [b]	Application deployment
GitHub	✓	✗	✗	✗	✓	✗	✗	✗	✓	✗	✗
Sourceforge	✓	✗	✓	✓	✓	✗	✗	✗	✓	✗	✗
GoogleCode	✗	✗	✓	✗	✓	✗	✗	✗	✓	✗	✗
CodePlex	✓	✗	✓	✓	✓	✗	✗	✗	✓	✗	✗
StackOverflow	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗
BlueMix	✗	✗	✓	✗	✗	✗	✓	✗	✗	✗	✓
Chef Supermarket	✗	✗	✗	✗	✗	✗	✓	✗	✓	✗	✗
LinkedIn	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗
geeklist	✓	✓	✗	✗	✗	✗	✗	✗	✓	✗	✗
Snipplr	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗
Masterbranch	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗
Dzone	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗
codeproject	✓	✗	✓	✗	✓	✗	✗	✗	✓	✗	✗
PaaSage SN	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓

[a] Features: follow and news feed

[b] User assistance based on data analysis of the repository

Table 2.1: Feature comparison

The lack of social networking features of GitHub came to fill the Geeklist platform [28], where developers and IT companies can discover and share the work they have done, connect with other companies in a social network manner or join development communities. Another code hosting platform is Snipplr [29], where developers can upload short code snippets but not full programs, in order to keep all of their frequently used code in one place that is accessible from any computer and any user. Masterbranch [30] is a new under development platform that allows collating and sharing of projects within a user's profile. This profile works similarly to LinkedIn and has an incentivisation scheme called DevScore, coupled with unlockable achievements that add a gamification element. Dzone [31] is essentially a link repository for developers allowing link sharing and incentivisation based on voting for the popular links. The Code project [32] website and forum allow code-specific discussion and share relevant articles and news, contains blogs, newsletter and a questions and answers section.

The above systems can be further classified based on whether they use a repository to store software-related information (code, models, configuration, or execution histories) and whether this information is shared and raised through crowd sourcing [33]. GitHub, GoogleCode, CodePlex, SourceForge, BlueMix, Chef supermarket, and our platform store at least one type of software-related information and all systems but BlueMix are raising shared content in their software-related repositories via crowd sourcing. Our social network platform is the only solution that analyzes information in its software-related repositories in order to provide users with assisting suggestions and hints. The platform targets the model of the application and the Software code of the application is not stored inside the social network platform.

2.2 Scalability in social networks

Arguably two of the largest existing networking platforms are LinkedIn and Facebook. The caching technologies of these networks are of great interest for the way they are managing and storing vast amounts of data.

LinkedIn, the largest professional network, stores hundreds of terabytes of data to Project Voldemort [34], a key-value store, inspired by Amazon Dynamo [35], another well-known key-value store. LinkedIn stores to Voldemort pre-computed offline data. For example, it stores the results of data mining applications, such as "People You May Know" feature, that are running on hundreds of terabytes to make an estimation and are using Hadoop as the computational component of those estimations. Voldemort and Dynamo have the same following requirements: (1) a simple *get/put* application interface (2) A *replication* factor, the number of replicas for each key-value tuble, implemented using vector clock, (3) a *required read* factor to succeed a get request and (4) a *required write* factor to succeed a put request.

Facebook, the largest social network, needs to handle large amounts of information every second. An overview of its architecture is presented at [36]. Facebook uses Linux Apache Memcached PHP (LAMP) website and communicate with other services and components to achieve scalability and performance. Facebook uses PHP, with an integrated compiler to optimize and turn PHP into C++ code. It continues use MySQL as a simple key value store without the relational aspect of the database. For the rendering of the web pages the BigPipe [37] is used.

Facebook serves billions of requests per second using memcached [38]. In this magnitude of scale, Facebook has several pools of memcached servers (regional pools) around the globe. A request for a single page can produce hundred of requests to the back-end system. Memcached is used to store not only key-value from MySQL queries but also pre-computed results from sophisticated algorithms. In order to achieve a near real time communication experience to the end user, memcached servers have to be efficient, reducing latency to minimum.

The research question in such systems is when a particular key will be invalidated. This problem occurs according to [38] in two cases: (1) *stale sets* and (2) *thundering herds*. A stale set occurs when a web server sets a value to the memcached that does not reflect the real value of the database. Thundering herds occur when a specific key has a heavy read and write activity at the same time. Stale sets are resolved by an N-bit token, that is bound to a specific key and sent from the memcached to the web server that wants to update the key when a cache miss occurs. If a delete request is received, the request for updating this value from the client is rejected. The thundering herbs are solved by configuring the memcached servers to return an N-bit token only once every ten seconds per key.

Our social networking platform, inspired by Facebook, integrates memcached in its back-end architecture and it is configured to properly interact with the caching application.

2.3 Configuration management and deployment

Another key component in a portfolio of DevOps tools is configuration management (CM) [39], the process of maintaining a detailed recording of software and hardware components in an infrastructure. An effective CM process provides significant benefits including reduced complexity through abstraction, greater flexibility, faster machine deployment, faster disaster recovery, etc. There are numerous configuration management tools from which a system administrator can choose, however the most widely known are: Bcfg2 [40], CFEngine [41], Chef [3] and Puppet [42]. Each of these tools has its strengths and weaknesses [43], [44]. In a DevOps environment, a CM solution is often combined with provisioning and deployment tooling [26].

The social network platform uses Chef as a CM and deployment automation tool to support professional network users, and SNP integrates the Chef cookbooks in its platform.

Furthermore, a recent trend in DevOps software development is continuous integration (CI) [45] and automated code deployment and testing off of online code repositories. Travis [46] is a CI tool that automatically detects when a commit has been made and pushed to a GitHub repository, subsequently tries to build the project, deploy and run tests, and notify the user of the status. Another popular CI tool is Jenkins [47], an open-source software tool for testing and reporting on isolated code changes in real time. Similar to Travis, Jenkins enables developers to find and solve defects in their code rapidly and automates the testing of their builds.

Although our social networking platform does not provide a complete CI solution because it is not a code hosting platform, but it automates the deployment of complex applications through a model-driven process.

2.4 Topic classification

Another technology that has been frequently used by networking platforms and we have integrated to our SNP is Natural Language Processing (NLP) [48]. Specifically, we focus on social networking usage of NLP and how knowledge can be mined from repositories of Q&A sites.

NLP has been used to process Twitter's messages and come to some results according to the classifications. Twitter has a good pool of micro-blog text which is suitable for NLP because of the small text sentences that users are allowed to post. Those posts describe emotions, feelings, opinions or situations. So several techniques [49] [50] [51] have been introduced to process and classify twitter posts in several categories.

StackOverflow (SO) is not left without NLP, because it can be seen as a repository of Q&A for programming questions. This means, that most of the questions and answers in SO contain some kind of a description at first and some code afterwards. An autoComment tool [52] is proposed using NLP which maps the code from developer projects and locates the same code somewhere in the SO Q&A, if it exists. If autoComment matches a segment of the developer's code with a code segment at SO, it performs NLP to the description of the code and inserts the modified description to the developer's code.

A trend in Social Networking sites is the ability of users to “tag” their posts. Those tags describe the users' goals and interests. Tagging SO questions involves askers selecting appropriate keywords to broadly identify the domains to which their questions are related. There also exist mechanisms by which other users can subscribe to tags, search via tags, mark tags as favorites, etc. This users' classification of context is used by PaaSage social network platform as described in Section 3.3.

The social network platform uses those tags and the Natural Language Processing to answer the following research questions:

- Can the platform identify the similarity of a given question with other questions already posted in the system.

- Can the platform map a question with a relevant query to the repository in order to provide the one who asks with an appropriate response.

- Can the platform paraphrase the queries according to the user's arbitrary input in order to meet the previous objective.

Chapter 3

Implementation

This chapter describes the implementation of our social networking site, the User Interface and how the system scales.

The system architecture is consisting of the following components, as shown in Figure 3.1:

- In the first layer, which is the front-end layer, lives (1) the social networking engine, which runs all the PHP scripts, as described in Section 3.1 and (2) the CDO client which is responsible for the communication with the repository of the application models, as described in Section 3.1.2.
- In the second layer, which is the caching layer, lives the Memcached caching system, which described in Section 3.2.2.
- In the third layer, which is the back-end of the system, lives the social network MySQL database, the CDO server, as well as the CDO repository of application models, which is a MySQL database.

In order to achieve the scalability of the system, two different system architectures are examined at the first two layers: (1) We added more than one social network engines at the first layer of the system. In this implementation, in order to keep the file system in consistent mode we integrated NFS server along with Apache Zookeeper [53] as described in Section 3.2.1. (2) We added more than one memcached nodes at the second layer in order to add more CPU capacity and improve the system's response time.

3.1 Social networking platform

The social networking platform is implemented over the extensible Elgg social network framework [54]. Elgg is an open source software written in PHP, that uses MySQL for data persistence and supports jQuery [55] for client-side scripting.

The overall architecture of Elgg Social Network is shown in Figure 3.2. The Elgg Social Network is structured following the key concepts of Model - View

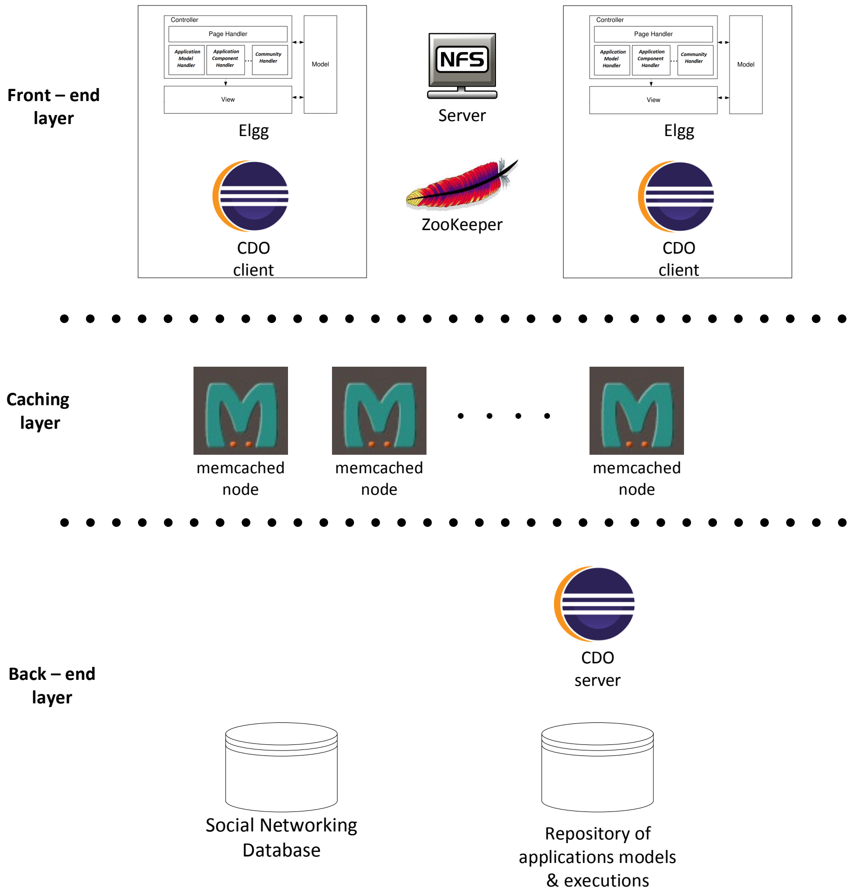


Figure 3.1: The overall social networking architecture

- Controller (MVC) known architectural pattern for User Interfaces. The MVC system is analyzed here, depicting the most important aspects of it from the Elgg's perspective. Explaining the Elgg's architecture by using MVC system will make the understanding of Elgg less complex.

Figure 3.2 shows the model, view, and controller parts of Elgg's architecture. In a typical scenario, a web client requests an HTML page (e.g., the description of an application model). The request arrives at the *Controller*, which confirms that the application exists and instructs *Model* to increase the view counter on the application model object. The controller dispatches the request to the appropriate handler (e.g., application model, component handler, community handler) which then turns the request to the view system. View pulls the information about the application model and creates the HTML page that is returned to the web client.

The **Model** of the framework is structured around the following key concepts as shown in Figure 3.3:

- *Entities*, classes capturing social networking concepts: users, communities, application models. Elgg Core comes with four basic objects: ElggObject, ElggUser, ElggGroup, ElggSite, ElggSession, ElggCache and a lot of other classes necessary for the proper engine operation.
- *Metadata* describing and extending entities (e.g., a response to a question, a review of an application model, etc.).
- *Relationships* connect two entities (e.g., user A is a friend of user B, user C is a contributor to an application model, etc.) and are persisted in the social network database.
- *Annotations* are pieces of simple data attached to an entity that allow users to leave ratings, or other relevant feedback.

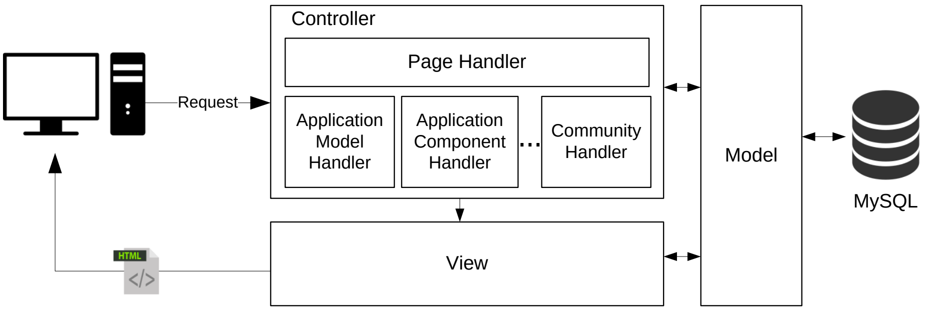


Figure 3.2: Architecture of the Elgg social networking engine

All Elgg objects inherit from ElggEntity, which provides the general attributes of an object. Elgg core comes with the following basic entities: ElggObject, ElggUser, ElggGroup, ElggSite, ElggSession, ElggCache, as well as other classes necessary for the operation of the engine.

The **controller** component of MVC model of Elgg consisting of the *Actions* of the system which are the primary way the users interact with the Elgg site. An action in Elgg Framework is the code that make changes to the database when a user performs an action like logging in, posting a comment, or creating an application model. The action script processes input, makes the appropriate modifications to the database, and provides feedback to the user about the action. By default, actions are only available to logged in users and include Cross-Site Request Forgery (CSRF) Security token to overcome session fixation [56], Session Hijacking [57] and Cross-site Scripting [58].

Additionally, the controller component includes the *Events* and the *Plugin Hooks*, which are used in Elgg Plugins to interact with the Elgg engine. Events and hooks are triggered at important times throughout Elgg's boot and execution process, and allows plugins to modify or cancel the default behaviour of Elgg.

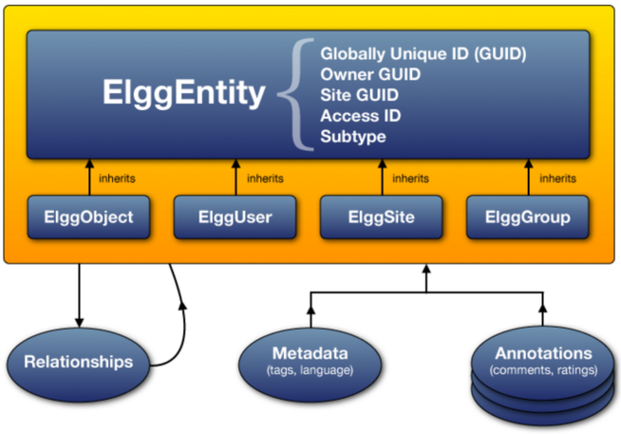


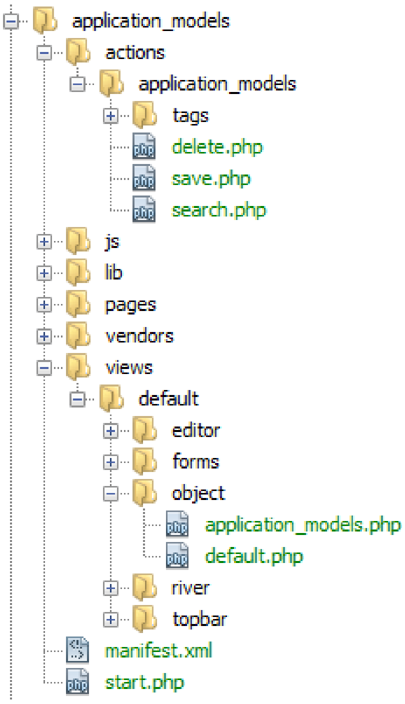
Figure 3.3: The Elgg engine data model

When an event is triggered, a set of handlers is executed in order of priority. Each handler passes the arguments and has the option to influence the process. When the execution of the current handler is completed, the “trigger” function returns a value based on the behaviour of the handlers.

The **View** component is responsible for creating the output. Generally, this will be HTML sent to a web browser, but it could also be XML, JSON or any other data formats. View handles everything from the layout of pages and chunks of presentation output (like a topbar) down to individual links and form inputs.

Elgg comprises a core system that can be extended through plugins (examples are the Cart system and the handling of Application Models). Plugins add new functionality, can customize aspects of the Elgg engine, or change the representation of pages. A plugin can create new objects (e.g., ApplicationObject) characterized (through inheritance of ElggEntity) by a numeric globally unique identifier (GUID), an owner GUID and an Access ID. Access ID encodes permissions ensuring that when a page requests data it will not touch any data the current user does not have permissions on. All plug-ins share a common structure of folders and PHP files, following the MVC model of Figure 3.2.

The hierarchy of a plug-in is shown in Figure 3.4. The folder *actions* includes the actions applied on application models. Every active participation of the user is performed via an action. Logging in, creating, updating or deleting content are all general categories of actions. The *views* folder contains the *php* forms applied on application models, *river* events (Elgg terminology for live feeds). Viewss are responsible for creating the output for the client browser. Generally, this will be HTML, but it can be also JSON or other format. *Pages* overrides elements of core Elgg pages and can be from chunks of presentation output (like sidebars) down to individual html code. The *js* and *lib* folder provides javascript and *php* library functions. Finally, the *vendors* folders include third-party frameworks such as Twitter's bootstrap front-end [59]. The most important file of a plug-in is the *start.php* script, which contains the *page handler*. Page handler is a function manages the plug-in pages enabling custom url redirect to a specific page. The plug-in initialization is also defined in the start.php and registers actions, events and determines the views.



```
application_models
    actions
        application_models
            tags
            delete.php
            save.php
            search.php
    js
    lib
    pages
    vendors
    views
        default
            editor
            forms
            object
                application_models.php
                default.php
            river
            topbar
    manifest.xml
    start.php
```

Figure 3.4: The structure of the application description plug-in

Finally, as mentioned before, for client side scripting the jQuery is used. The main reason why jQuery is preferred in this work over of pure JavaScript [60] is that it's a light library which pushes content to the client machine, it therefore reduces the wait time for server response. Plus, it's smaller than Flash, so it results in smoother playbacks and less errors. Furthermore, jQuery works anywhere since is cross-browser compatible with any browser, mobile phone or tablet, and Apple devices. Finally, another hand-solving advantage of jQuery is its simple syntax.It is designed to make it easier to navigate in a document, select HTML DOM elements, create animations, handle events, and developing Ajax applications.

Thus, the jQuery is used by SNP for implementing client side scripting and for the remaining of this chapter, when we refer to JavaScript, we actually refer to the jQuery library. Furthermore, some other JavaScript libraries are used in order to make the User Interface more powerful. One of those libraries is the Chart.js [61] library which is used to generate the graphs and charts in execution's page.

3.1.1 Extending the core Elgg platform

As described in the previous section, the new functionality of Elgg social networking platform can be introduced by new plugins. Since the modification of the core system is not a good practice, because it makes the system more difficult to implement and does not let it upgrade to the new versions of Elgg framework. The following plugins are implemented:

ApplicationModel. The ApplicationModel plugin has a *page handler* to manage the application Model pages. Also, ApplicationModel has client side JavaScript for manipulating User Interaction and dynamic pages. Furthermore, some php libraries are implemented, for example a library for interaction with CDO client or a library for manipulating Application models.

Components. The Components plugin has a *page handler* to manage the Components pages and their Categories and a PHP library to interact with Chef supermarket.

CustomView. The CustomView plugin has all the necessary customization of the PaaSage social networking platform (PSNP). All custom views of the system are implemented in this plugin. This plugin overrides all the default views of the Elgg that should be changed and contains client side JavaScript. Furthermore, CustomView has the following seven page handlers: p̂rofile responsible for profile pages, âvatar responsible for the photos of user pages, ŝettings responsible for the pages of user settings, f̂riends responsible for the friends of the user pages, ĉontact responsible for the Contact Information of the PSNP, r̂eview responsible for the reviews of Application Models and ŝearch responsible for the main search facility of PSNP. Finally, CustomView has all the required *Actions* of the plugin such as the vote up or down, the action to add a review etc.

NotificationSystem. This plugin is responsible for the notifications of the Social Network which contains a relevant page handler, JavaScript for client side scripting and a php server side library.

Tags. This plugin does not include any page handlers but only the necessary actions for the Tags such as *add* or *delete* a Tag and a php library responsible for those.

UserStatistics. This plugin is responsible for collecting and displaying the information about the Users.

Memcached. This plugin has all the essential functionality for memcached implementation as will be described in Section 3.2.2.

ZookeeperRecipes. This plugin has all the essential functionality for memcached implementation as will be described in Section 3.2.1.

Groups. The Groups plugin is the default plugin of Elgg Framework modified to support the required functionality.

Messages. The Messages plugin is the default plugin of Elgg Framework modified to support the required functionality.

Twitter bootstraping of Elgg

Responsive web design (RWD) [62] is a web design approach aiming at crafting web application sites to provide an optimal viewing experience, which it provides easy reading and navigation with a minimum of resizing, panning, and scrolling, across a wide range of devices (from mobile phones to desktop computer monitors). However, the default CSS of Elgg, which is part of the View component of architecture, does not offer responsive web pages. Thus the ideal solution is to integrate Twitter Bootstrap to Elgg viewing system.

Twitter Bootstrap [63] [64] is a free and open-source collection of tools for creating dynamic websites and web applications. It contains HTML and CSS-based design templates for typography, forms, buttons, navigation and other interface components, as well as optional JavaScript extensions. It aims to ease the development of dynamic websites and web applications.

We customize the Elgg view inserting the Twitter Bootstrap view system. The default view system of Elgg changed to support the Twitter Bootstrap responsive grid.

3.1.2 Communicating with CDO server

As mentioned in 1.1, the execution history of deployments of application models and the description of those models are stored in the CAMEL information repository, which is implemented as an Eclipse CDO server. In order to communicate with the CDO server, a CDO Java client is needed. The CDO client stands between the social networking engine (Elgg server) and the CDO server, making the exchange of information between those two possible, as shown in Figure 3.1.

Specifically, regarding the communication between the CDO server and the client, the CDO client opens one or more sessions to the CDO server. Each session represents a connection to the CDO repository and provides a broad API to interact with it. A session does not provide direct access to model instances; views or transactions are needed in order to navigate or modify the model instance graph. The implemented CDO client exposes read/write access to the repository for either viewing the execution histories or the model of the applications, or for storing new execution models. For the communication between the social networking engine and the CDO client, the CDO client exposes a RESTful API to the social networking engine providing all the necessary methods.

For example, when a user from the social network platform requests the execution histories of an application, the engine sends a request to the CDO client through the RESTful API, the CDO client receives the request and forwards an appropriate request to the CDO server. The CDO server receives the request and queries the repository of application models and executions. When the CDO server receives the response, it forwards the response back to the CDO client, which forwards the response back to the social networking engine. The social networking engine transforms the response to JSON format, in order for it to be readable by the JavaScript. JavaScript plays the final role, by projecting the execution histories in a proper table to the end user that requested the page of the executions of an application.

As a future work, message queue techniques can be used for the communication between the CDO client and the social networking engine. One message queue technique is ZeroMQ [65] which is a high-performance asynchronous messaging library, aimed at use in scalable distributed applications. Another well-known message queue system is RabbitMQ [66], which is a messaging broker and provides a common platform to send and receive messages. Those techniques achieve communication between different language components of an application.

3.2 Scalability

A lot of work has been made on the scalability of social networking platforms. The scale of social networks is very important for platforms such as Facebook which has one billion users. We integrate known practises to make the social networking platform scalable and efficient. From where the Elgg social networking engine was implemented to run in only one single machine, in this work, we distribute and design a scalable architecture. We replicated the social networking engine and we introduced caching techniques into our architecture.

3.2.1 Social networking engine

This section describes how the horizontal scale of social network engine is achieved by adding more than one Elgg servers. At the layer 1 of Figure 3.2 lives the Apache2 server, which as the stress test of the system indicates in Chapter 4, takes a heavy load on CPU utilization.

The heavy load of Apache2 server occurs by the nature of Elgg framework. Because the Elgg core system is implemented to be extensible and configurable, every time a simple page or just an AJAX call is received by the Elgg, the Elgg framework performs the following heavy tasks: it broadcasts an *init system* event; this event is caught by all plugins of Elgg and at this initialization phase the plugins register: (1) the page handlers, (2) the PHP libraries, (3) the actions, (4) the events and hooks, (5) the JavaScript libraries and (6) the CSS scripts. Therefore, all those actions generate a heavy load resulting in consuming CPU utilization and slowing the response time of the system. We introduce more than one Apache2 servers running the social networking engine of Elgg framework to overcome this.

The social networking engine keeps some information in the file system instead of in the social network database. This information includes the profile photos of users and any other images such as photos that users add to the community groups. Furthermore, the initial configuration of social networking engine keeps in the file system some caching files. Those files represent some views of the web pages which are independent of any specific users and remain unchanged among all users. This file system caching feature is removed from the social network engine because it is more efficient to use memcached for the caching instead of the slow file system.

In order to all social networking engines have access to the same file system, the Network File System (NFS) [67] is used. NFS allows a server to share directories and files with clients over a network. With NFS, users and programs can access files on remote systems as if they were stored locally.

In our implementation, NFS is configured and used in order to allow all social networking engines to gain access to the same file system store. An NFS server is installed in one of the SN engines and all the other SN engines have an NFS client accessing the remote file system.

Distributing social network engine was not an easy problem to solve, so Apache ZooKeeper [53] is used in order to achieve synchronization. Apache ZooKeeper [68] is a service for coordinating processes of distributed applications. Since ZooKeeper is a part of a critical infrastructure, it aims to provide a simple and high performance kernel for building more complex coordination primitives at the client. We use this service in order to enable highly reliable distributed coordination among the file system and the social networking engine. Apache ZooKeeper provides a tree abstraction where every node in that tree (or znode) is a file on which a variety of simple operations can be performed. ZooKeeper orders operations on znodes so that they occur atomically. Therefore there is no need to use complex locking protocols to ensure that only one process can access a znode at a time. The tree represents a hierarchical namespace, so that many distinct distributed systems can use a single ZooKeeper instance without worrying about their files.

Social networking engine uses Apache ZooKeeper in order to keep the file system consistent, in rare but possible scenarios like two users trying to upload a file to the same group simultaneously. When a social networking engine wants to write a file in file system, it first locks the specific path and after finishing the write operation it releases the lock. Thus, a file can never be corrupted.

For communication between the Apache ZooKeeper and the Elgg framework, the php-zookeeper-recipes [69] are used by the ZookeeperRecipes plugin. Specifically, the *exclusive locks* of Zookeeper are used to keep the system in consistent mode.

3.2.2 Memcached

This section describes how we integrated memcached [70] to our system architecture. Memcached is an open source, high-performance, distributed memory object caching system. We chose memcached, because it is a generic simple in-memory key-value store. It has a powerful API available for PHP. After memcached integration the system decreased its response time and its performance.

Memcached is added in layer 2 of the system architecture as shown in Figure 3.1 and is used for storing the key-value tuples. The following data are stored in memcached:

1. MySQL responses, which are values from social network database such as entities of social network engine, applications, components, users, group discussions. By storing those values, there is no need to query the social network database, but SN Engine is getting the key-values directly from memcached.

2. Views of the web pages which are independent of any specific users and remain unchanged among all users, and werte previously were stored in the file system.

3. JavaScript code results. Some JavaScript code is time consuming to be generated. For example, PHP sends the execution data to JavaScript and then it iterates the data in order to generate the tables and the graphs of execution histories.

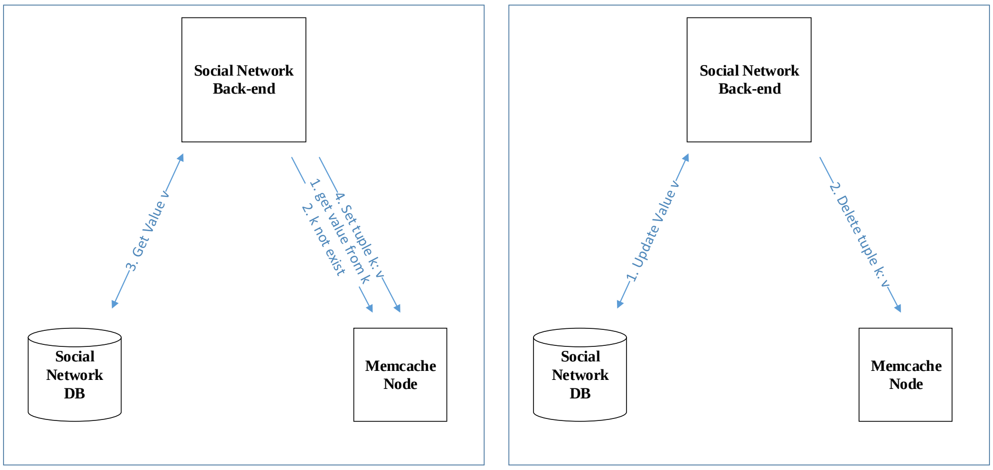
4. Executions histories from repository of application models. By storing the executions of applications at memcached, the system's response time decreases. That is due to the PHP modules not needing to go through the heavy CDO client-server communication but them getting the executions of applications directly from memcached.

The keys stored in memcached must be unique. This is implemented using as a prefix of the key the globally unique identifier (GUID), generated by Elgg, and a string value which is describing the data. So, the execution data of an application model with QUID 1000 is stored in memcached with key *1000:execution-data* and the value is the json representation of those data. All tuples at memcached are inserted with the maximum key expiration time of thirty days.

The basic actions of memcached are the get, set and delete of a tuple, as shown in Figure 3.5. In the first scenario, when a value from SNP is requested for a particular key *k*, first a query will be send to memcached requesting the tuple with that key *k*, if *k* not found or it has expired, the SNP will request that key from the social network database. Finally, when SNP gets in touch with the data of the key, it will send those data to be cached in memcached, as shown in the Figure 3.5a. In the other scenario, when a value in the SNP is updated, the memcached key will be deleted(invalidated) as shown in the Figure 3.5b.

The Elgg framework comes with the potential use of memcached but is restricted to the fact that a memcached node must be in the same machine as the Elgg framework. Plus, it makes it more difficult to configure when an insertion in memcached node will take place. Therefore, a new Elgg plugin is implemented called **Memcached** using the memcached PHP library [71]. The basic memcached functions offered by this plugin are: (1) Add memcached nodes, (2) add a key to a node, (3) get a value of a key and (4) delete a key-value item.

The Apache JMeter [72] was used to measure the response time of the system and the sysstat tool [73] was used to measure the CPU usage. Section 4.1.1 shows the performance results of this implementation.



Scenario a)
(1) Request a key k from Memcached node, (2) the k not found in memcached and the SN back-end (3) requests the value v from SN DB. When the SN back-end receive the v, (4) stores it to memcached node

Scenario b)
When (1) a value updated to SN DB, (2)the corresponding memcached key will invalidated.

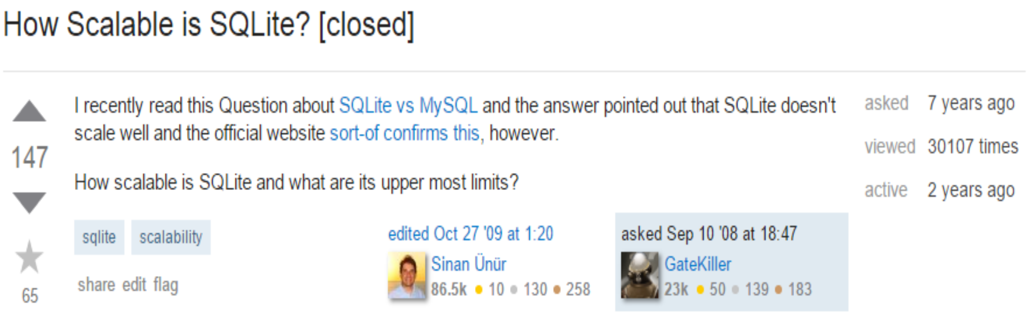
Figure 3.5: The scenario (a) depicts a request from memcached when the key does not exist and scenario (b) depicts an updated operation of a value

3.3 Topic classification

Topic classification, using Natural Language Processing (NLP) [48], is a feature added to the social network platform. NLP is used in the interactions between the users and the SNP, in the way the platform can understand and determine the type of user input and provide helpful possible answers.

For example, one real question from the StackOverflow is shown in Figure 3.6, which asks the community about the scalability of SQLite and MySQL. From this specific question several important outcomes can be mined. First, this question is very difficult to be answered without using targeted analyses of execution histories of those applications. Second, similar questions can be asked to our own social networking community and would be helpful to use it as a training set to our topic classification tool presented in this section. Finally, many questions of this type exist in the StackOverflow community and we would like to use them in realizing topic classification.

As the Figure 3.7 shows, a user of SNP poses a question to our platform. The platform can process the user's input and classify it into: (1) a similar question that exists in our community or (2) an automatically generated query to the CAMEL repository. In the first scenario, a list of answers from the pre-existing question can be provided as feedback to the new question. In the second scenario, the question can be further processed and a query to CAMEL repository be automatically generated. An automated mechanism can be implement for the second scenario but this is out of the scope of this thesis.



How Scalable is SQLite? [closed]

147

65

I recently read this Question about SQLite vs MySQL and the answer pointed out that SQLite doesn't scale well and the official website sort-of confirms this, however.

How scalable is SQLite and what are its upper most limits?

sqlite scalability

share edit flag

edited Oct 27 '09 at 1:20
Sinan Ünür
86.5k ● 10 ● 130 ● 258

asked Sep 10 '08 at 18:47
GateKiller
23k ● 50 ● 139 ● 183

asked 7 years ago
viewed 30107 times
active 2 years ago

Figure 3.6: A real example question from StackOverflow

Our implementation lies in between those two scenarios. Some queries to the CAMEL repository are implemented and if a question is asked that in a way resembles a specific query, the platform will provide this query's results as an answer to the user's question. Furthermore, if this question has a similar question and that last question has an answer from a sophisticated user, who had created an hand-crafted query to the repository and had provided the results back to the user, those results can be used as a feedback to the new question.

We added the Naivy Bayes Natural Language Understanding algorithm to the SN using the Natural framework [74], which is implemented with node.js. In general, machine learning algorithms such as Naivy Bayes require an input of training data, called data set. This training data is pulled from questions users post in StackOverflow (SO). Those questions are an excellent repository to train the NB algorithm, because they are categorised by tags (this helps our classifier) and are real questions that our social network community may be interested. This was also a necessity since by the time that this implementation took place, the social network platform did not include a sufficiently large amount of questions in its repository.

In general, the main actions that SO users is shown in Figure 3.8. When users ask questions in SO, they must specify some tags describing their questions. A tag is a keyword or a label that places their question in a category with other, similar questions. SO users usually try to add as many tags as possible in order to make their questions popular and get them answered. SO permits users to add up to only five tags in each of their questions. After a question is posed, the whole community of SO can vote the question up or down and the privileged users can flag the question as *duplicate*, *off-topic*, *unclear*, *too broad* or *primarily opinion-based*. This way, low quality questions will be removed from the site resulting in keeping the questions repository clear and helpful for other potential developers to use.

For the training sets, the data to train the NB algorithm, the most voted questions of the SO community are used. Those questions have emerged as the best questions in their field and surely, we avoid the case of using a training set with miss-tagged questions that would result a miss-guided NLP classification. The

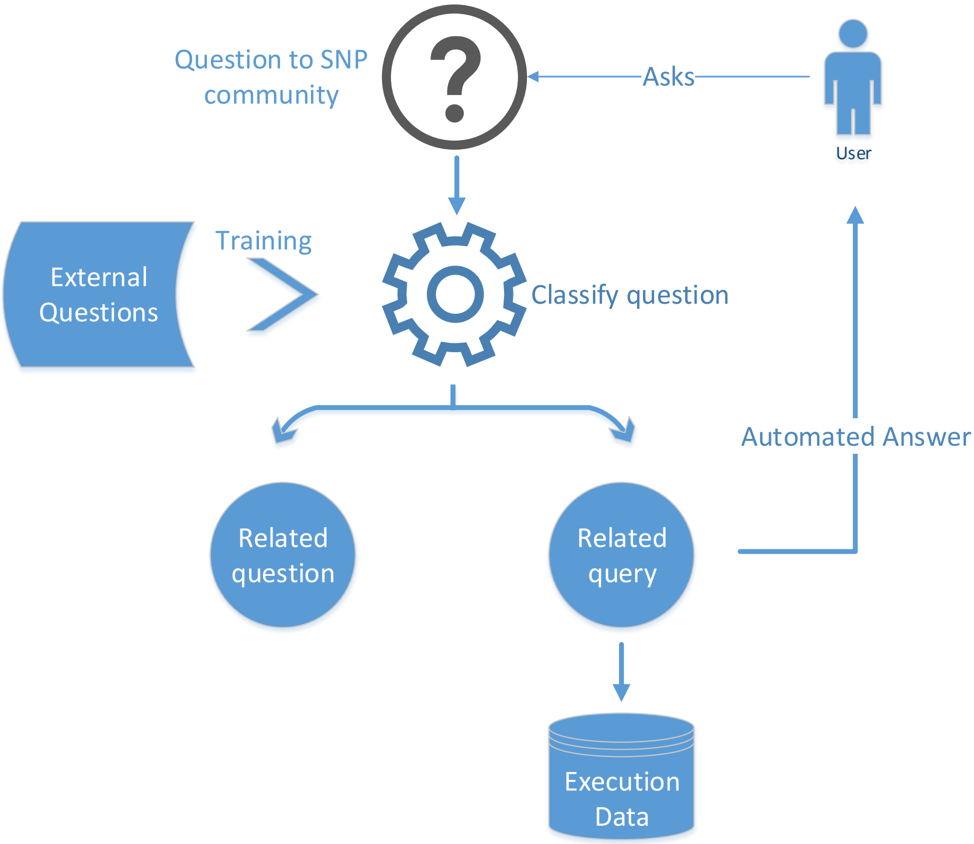


Figure 3.7: Classification of questions and automated feedback to user

NLP training set is retrieved from SO site using the stack exchange(SE) API [75]. The SE API is a powerful API, which allows us to take the questions, the answers, the users and all the information that exists in SO site through a programming interface.

The first training set of our Natural Processing Tool consisted of five tags, relative to our platform. Those tags were: *scalability*, *reliability*, *design*, *performance* and *optimization* and we got thirty questions per tag. For each of the tags, the thirty most up-voted questions from StackOverflow were retrieved and classified to each specific tag. Those exact tags, after classification, are transformed in classes in NLP classification, as shows the Figure 3.8

Every time a user asks a question to one of the platform's communities, the classifier determines the class of the question. After the class is found, and if the platform is able to determine a heuristic answer, it will post then the answer to the user's question automatically. All the users of the platform can vote this answer up or down, depending on its accuracy, or provide their own answers.

The second training set of Natural Processing Tool was retrieved by automat-

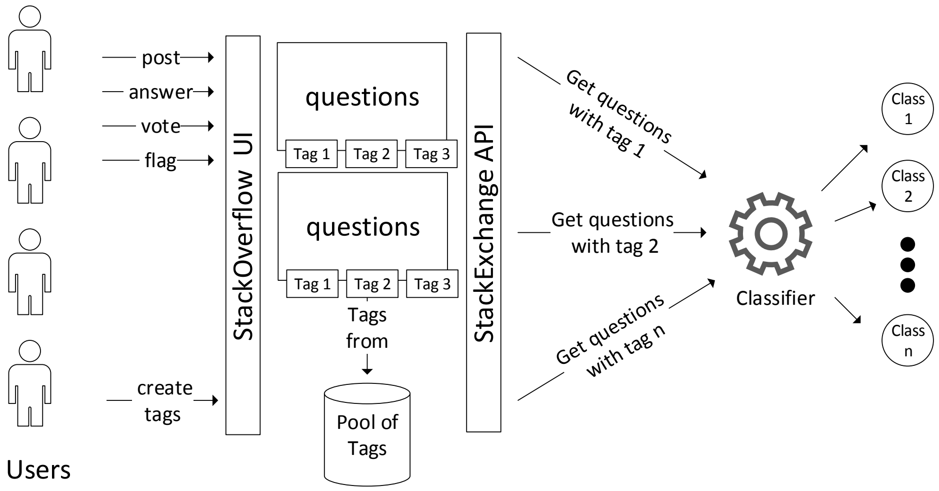


Figure 3.8: The main StackOverflow users' actions and the topic classifier

ically discovering tags. NB is trained with 10,000 questions from StackOverflow Q&A site, fifty questions per each discovered tag. The general algorithm is shown in Table 3.1. Firstly, the algorithm starts with a tag that is relevant to the social network platform such as *scalability* at line 01. Afterwards, using the SE API the algorithm gets the 5 most voted questions tagged with *scalability*. For each question (line 04), the populateClassifier clears the body from any html tags inserted by the StackOverflow users to beatify their questions (line 05) and classifies this question's body with each tag. Automatically, the populateClassifier proceeds to the next tag of this question. When the populateClassifier is finished the NB is trained. It should be noted that a question may have more than one tags, so a question can be classified to up to five tags / classes. Changing the threshold parameter at the following algorithm, the *populateClassifier* can classify questions with an arbitrary number of tags. At the following section the process to Bayes classification is described in more details.

3.3.1 Bayes classification algorithm

This section describes the Bayes classification algorithm. As the above code snippet shows at line 08, the algorithm classifies a document named *body* into the class *t*. This *body* is the content of the question that will be classified into the class *t*, which is the tag of the question. Diving in this function, the *body* is transformed to lower case and the Porter Stemming Algorithm [76] is used for suffix stripping, so the plural form of the words and the suffixes are removed (such as *-ing* and *s*). For example, the following words: *connected*, *connection*, *connections*, *connecting*, *connectionless* will all be transformed to the single word "connect". The Porter Stemming Algorithm does not use any dictionary but a simple list of suffixes. This

```
var tags = ['scalability']
populateClassifier(0)
function populateClassifier(i) {
  var qs = stackexchange.api.getQuestionsByTag(tags[i])
  foreach(qs as q)
    body = clear(q.body)
    foreach(q.tags as t)
      classify(body, t)
      if(not tags.exist(t) and tags.length() < threshold)
        tags.push(t)
        populateClassifier(tags.indexOf(t))
}
```

Table 3.1: Training algorithm

practise, makes the algorithm fast (10.000 different words in 8.1 seconds). After this process a table of words of this *body* is kept.

After the populateClassifier has finished, the *trainClassifier* is called (for simplicity the classifier is not shown in the above code snippet). The objective of *trainClassifier* is to make the document body ready for Bayes Classification. The purpose of *trainClassifier* is to count the number of occurrences of each word in each class.

After the above process is done, the classifier is ready, so when a future request for classification comes, the classifier returns the probability of a document being part of a class. This probability is calculated with the following formula:

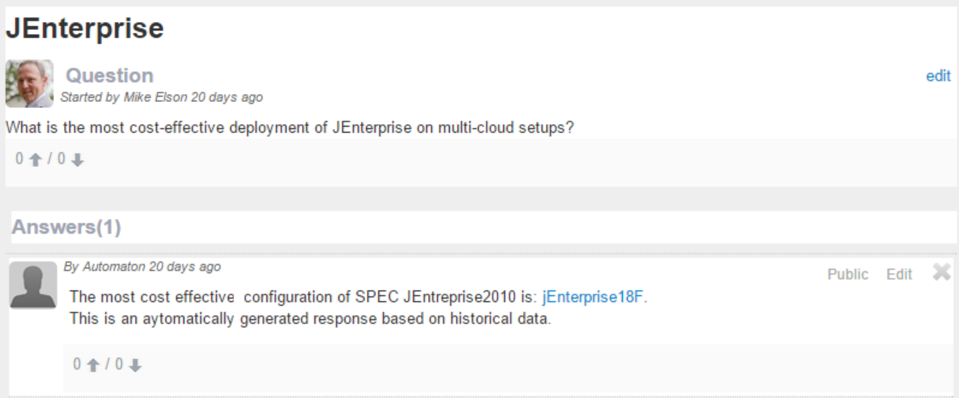
$$prob(d/c) = log\left(\frac{countedTerms(d,c)}{totalsTerms(c)}\right)$$

Where the probability of a document *d* to be in a class *c* is the logarithmic value of the division of the words(terms) of *d* found in class *c* by the total number of terms in *c*.

3.3.2 Automated answers

As described in Section 3.3 Natural Language Processing is used to determine the users' input question in groups.

When a user asks a question, the body of the question classified into categories and when the NLP classifies the question to a specific category, an approximate answer can be given in response. As the example shows in Figure 3.9, after the user's question, the classifier process the body and if the question is about the *JEnterprise* and the *cost effectivess* the approxiame answer "The most cost effectiveness configuration of SPEC JEntreprise2010 is: jEnterprise18F ..." is given. The users of the PaaSage social network platform can vote up or down the answer



JEnterprise

Question

Started by Mike Elson 20 days ago

What is the most cost-effective deployment of JEnterprise on multi-cloud setups?

Answers(1)

By Automaton 20 days ago

The most cost effective configuration of SPEC JEntreprise2010 is: jEnterprise18F.
This is an aytomatically generated response based on historical data.

Figure 3.9: Automated answer to user's question

and/or provide their own answers.

The automated replies provide the user with a direct possible answer to the question. So, it reduces the time waiting for an answer and provides a first helping hint about the user's question. If the answer is not efficient, other potential user of SNP can provide answers to the question and down-vote the automated reply.

Finally, a relevant question can be asked to the platform as "Which deployments of JEnterprise provide the best performance for the lowest cost in a multi-cloud setup?". This question is a paraphrased question of the previous one, talking about the same thing. The platform can find this similarity and provide the same automated answer.

3.4 User interface

This part of the thesis describes the User Interface of the social network platform that is implemented based on 104 mock-ups created by HCI expert team. A mockup is a realistic representation of what the product will look like, in our case the social network platform. The design of look & feel of SNP is made by HCI expert team in order to follow the modern trends in Web Applications design. In order to support those look & feel and the functionality of those mock-ups 25K lines of php, js and css code is written. The key design objective of the social network platform is to create a strong bond between (i) software engineering services for managing and deploying cloud-targeted application models; and (ii) community-oriented facilities for communication and collaboration between users. The interconnections between the two in the design of the user interface are depicted in Figure 3.10. The prototype implementation is publicly accessible on-line at http://socialnetwork.paasage.eu.

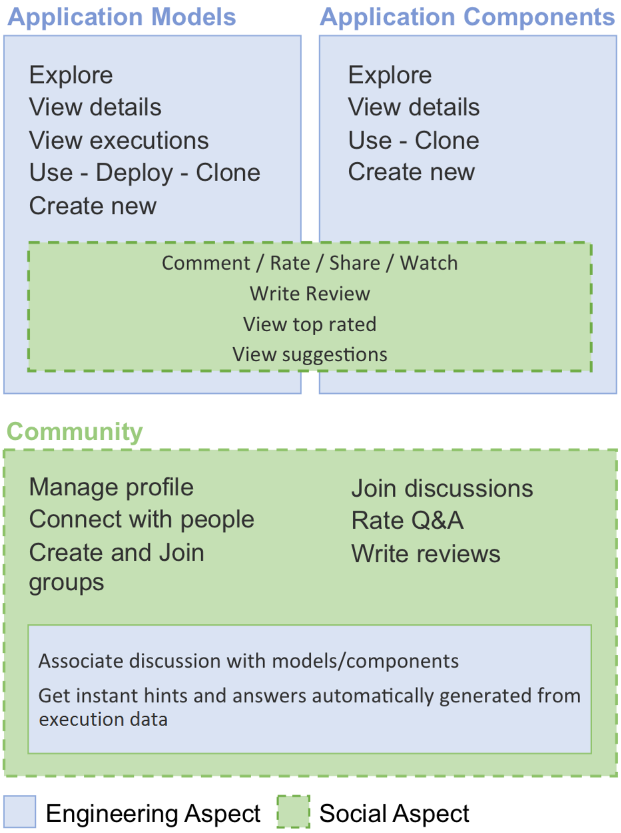


Figure 3.10: Engineering and social activities are seamlessly interweaved within the social networking platform

3.4.1 Design principles

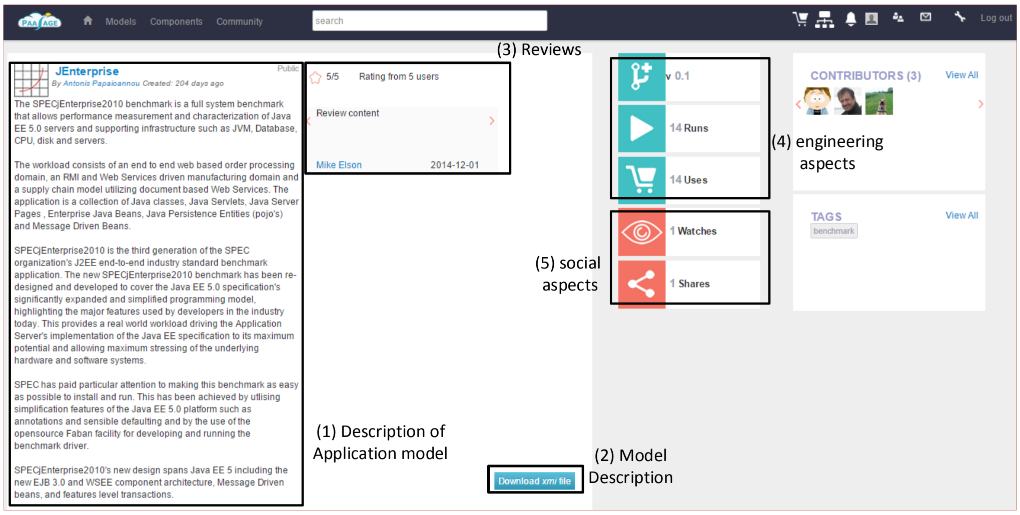
The discrete entities, which bind together the Social Networking with the application model aspects of platform are:

- *Application Models*. Application Models is a key entity of the platform. An example is shown if Figure 3.11, consisting of a human friendly description (label 1 in fig.3.11), the Camel Description of the model (label 2 in fig.3.11), reviews about the model (label 3 in fig.3.11). An overview of engineering aspects such as version and runs (label 4 in fig.3.11) and an overview of social aspects such as share and watch (label 5 in fig.3.11). The *share* action broadcast the model to the friends of the user that shares the model. The *watch* action notifies the user for future updates of the application model.

- *Components*. We have integrated the Chef supermarket components into social network platform. The components help the DevOps users to generate their application models as described in 3.5.1.

- *Users*. Users who basically are cloud deployment specialists and other users who want to know which deployment configuration they should use. Users can exchange knowledge to groups and benefit from the CAMEL repository. They can create or join groups, ask and answer questions, follow application models and create their own network of friends.

- *Groups*. As mentioned, every user of PaaSage social network platform can create or join groups. Groups help users to interact with each other and gain knowledge from experts.



JEnterprise

By Antonis Papaioannou Created: 204 days ago

The SPECjEnterprise2010 benchmark is a full system benchmark that allows performance measurement and characterization of Java EE 5.0 servers and supporting infrastructure such as JVM, Database, CPU, disk and servers.

The workload consists of an end to end web based order processing domain, an RMI and Web Services driven manufacturing domain and a supply chain model utilizing document based Web Services. The application is a collection of Java classes, Java Servlets, Java Server Pages , Enterprise Java Beans, Java Persistence Entities (pojo's) and Message Driven Beans.

SPECjEnterprise2010 is the third generation of the SPEC organization's J2EE end-to-end industry standard benchmark application. The new SPECjEnterprise2010 benchmark has been re-designed and developed to cover the Java EE 5.0 specification's significantly expanded and simplified programming model, highlighting the major features used by developers in the industry today. This provides a real world workload driving the Application Server's implementation of the Java EE specification to its maximum potential and allowing maximum stressing of the underlying hardware and software systems.

SPEC has paid particular attention to making this benchmark as easy as possible to install and run. This has been achieved by utlising simplification features of the Java EE 5.0 platform such as annotations and sensible defaulting and by the use of the opensource Faban facility for developing and running the benchmark driver.

SPECjEnterprise2010's new design spans Java EE 5 including the new EJB 3.0 and WSEE component architecture, Message Driven beans, and features level transactions.

5/5 Rating from 5 users

Review content

Mike Elson 2014-12-01

Figure 3.11: The application model home page

Gamification

Following recent trends in social networks design and with the aim to motivate users active and regular participation in the professional network, the design employs gamification features, meaning the use of video game elements in order to improve user experience and user engagement in non-game services and applications [77]. One gamification feature in the social network design is the reward system for active community members. As users contribute content (models, components, ratings, reviews, questions, or answers) they receive experience points leading to special badges visible to all community members. Other features are the Profile completeness bar with suggestions on how to increase it. Finally, the concept of Model badges awarded to application and component models in case of excelling performance. Badges can serve among others as goal-setting devices, status symbols, and indications of reputation assessment procedures [78].

3.5 Generating application models

The DevOps users of PaaSage social network platform can benefit from the automated creation of CAMEL baseline models presented at Section 3.5.1 or upload their own created models using external editors like EMF [12] tree based editor or GMF [79] editor presented at Section 3.5.2.

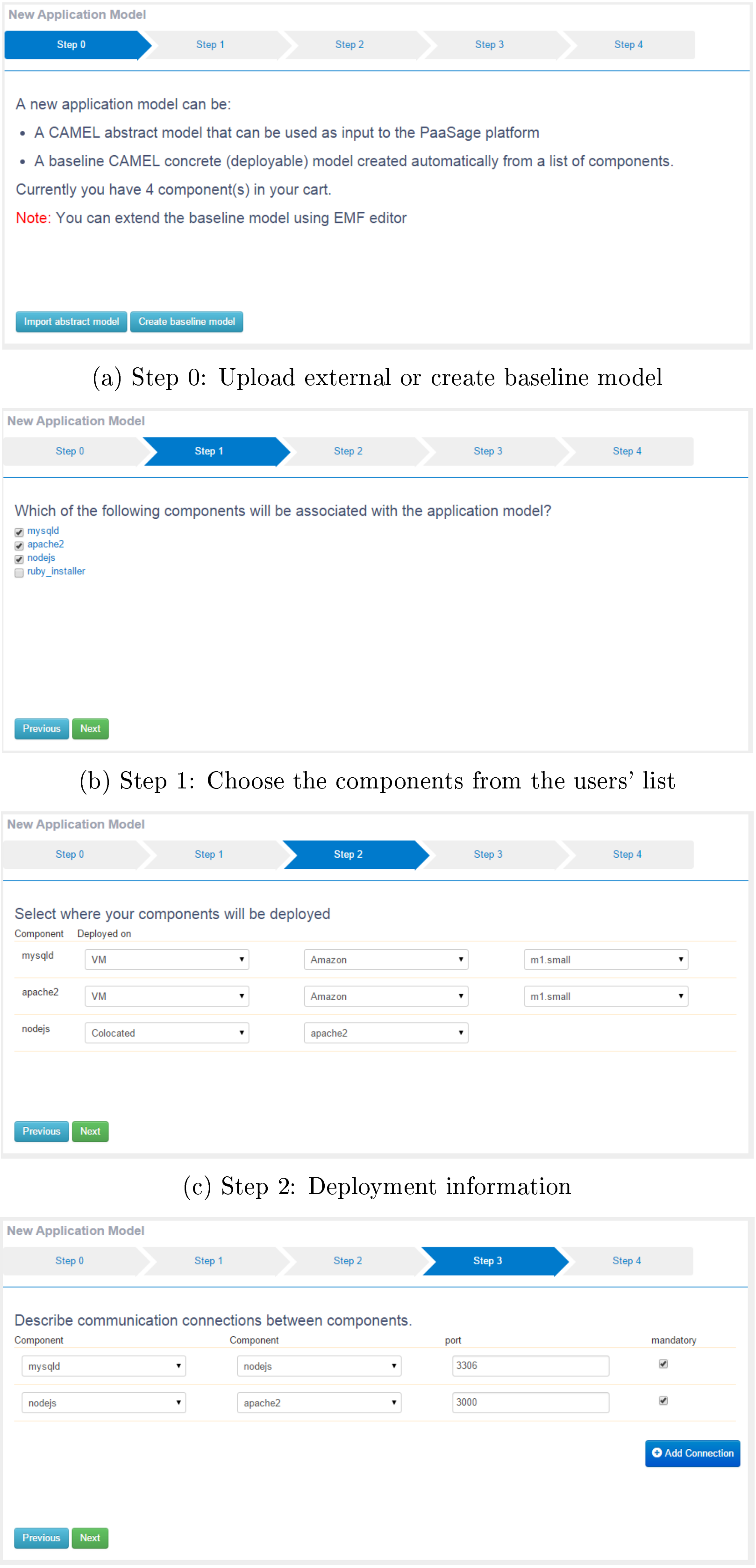
3.5.1 Baseline models

In order for users to create automated generated baseline CAMEL models, they can browse around the integrated Chef components inside the social networking platform and find the appropriate components for their applications. The platform has integrate a pointer for each Chef cookbook to the social network database using the Chef Supermarket API [80]. A PHP command has implemented in order to iterated through all Chef cookbooks and update the repository of SNP. This command has been configured and runs one time every day.

Through the application model creation Page of the PaaSage social network platform, the users can upload an external CAMEL model description of their application or create a new one with the help of the platform in 4 simple steps as shown in Figure 3.12. In the step zero 3.12a, the user is asked whether the new Application model already has an CAMEL model or whether the user wants to create a new application CAMEL model through automated generation. For automated generation, the user must previously put the Chef cookbooks that he/she wants in the user's cart. The selected Chef cookbooks will be shown as a list of components. Then, in step one 3.12b, the user selects from this list which of the components will be included in the application model. In the example shown, the user has four components *mysqld*, *apache2*, *nodejs* and *ruby_installer* and selects three of them. In the next step, shown in Figure 3.12c, the user provides the deployment information (to which cloud provider the Application will run and which type of VMs will be used). Also, some components can be collocated in the same VM. In this example, the *nodejs* component will be collocated in the same VM as the *apache2*. In step tree, shown in Figure 3.12d, the user provides the communication information between the components, for example the *nodejs* communicates with *mysqld* in the default mysql port *3306*.

In the final step as shown in Figure 3.13, the user provides the final needed information about the name of the Application model, a human friendly description and the version of the model. In the bottom of the form of the Figure 3.13 the user can find three actions: the *Previous* action, the *Save as draft* action and the *Finalize* action. The *Previous* action can performed all around the steps and make the user easily walk around the steps, giving him/her the opportunity to alter an option. The *Save as draft* action creates the CAMEL model of the application but the model is not publicly available and only the creator or the contributors of the Application can see or edit the Model. The *Finalize* action creates the CAMEL model of the application and makes it publicly available to the SNP users.

For the creation of the base line CAMEL model an AJAX request is sent to a Java tool, that automatically creates the application model according to the information that the user has already provided. When the Java tool generates the CAMEL model, it sends it back to the SNP. So the SNP can store this model to the repository. This tool is presented at [81].



New Application Model

A new application model can be:

- A CAMEL abstract model that can be used as input to the PaaSage platform
- A baseline CAMEL concrete (deployable) model created automatically from a list of components.

Currently you have 4 component(s) in your cart.

Note: You can extend the baseline model using EMF editor

(a) Step 0: Upload external or create baseline model

New Application Model

Which of the following components will be associated with the application model?

☑ mysqld
☑ apache2
☑ nodejs
□ ruby_installer

(b) Step 1: Choose the components from the users' list

New Application Model

Select where your components will be deployed

Component	Deployed on		
mysqld	VM	Amazon	m1.small
apache2	VM	Amazon	m1.small
nodejs	Colocated	apache2	

(c) Step 2: Deployment information

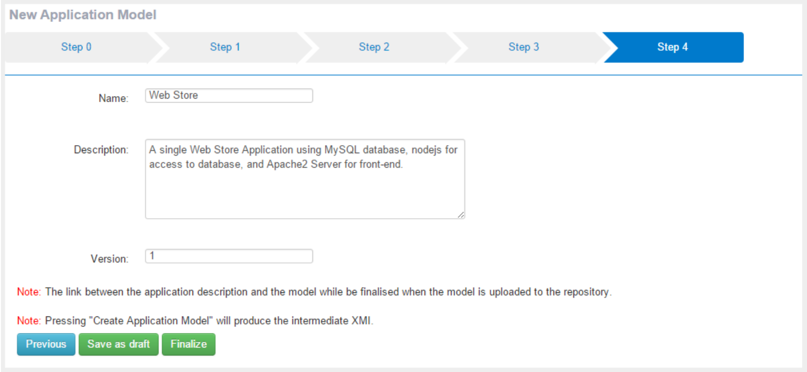
New Application Model

Describe communication connections between components.

Component	Component	port	mandatory
mysqld	nodejs	3306	☑
nodejs	apache2	3000	☑

(d) Step 3: Communication information

Figure 3.12: Steps for automated creation of baseline model



Name: Web Store

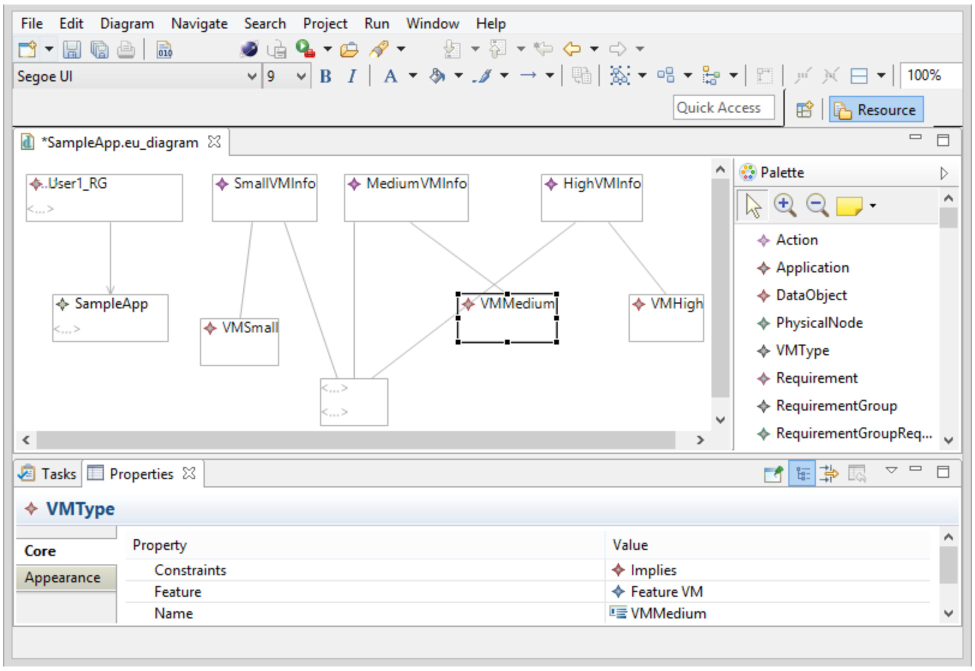
Description: A single Web Store Application using MySQL database, nodejs for access to database, and Apache2 Server for front-end.

Version: 1

Note: The link between the application description and the model while be finalised when the model is uploaded to the repository.

Note: Pressing "Create Application Model" will produce the intermediate XMI.

Figure 3.13: Final step of automated creation of baseline model.



Property	Value
Constraints	Implies
Feature	Feature VM
Name	VMMedium

Figure 3.14: GMF editor composition of a sample application

3.5.2 Graphical modeling

Advanced users of the PaaSage social network platform can compose application models through Graphical Modeling Framework (GMF) [79] which is an external Eclipse editor. GMF provides a set of generative components and runtime infrastructures for developing graphical editors based on Eclipse Modeling Framework (EMF) and Graphical Editing Framework (GEF). The GMF editor is generated from CAMEL *ecore* schema and provides the graphical palette to compose applications.

Figure 3.14 shows the composition of a sample application model with the GMF editor. The palette in the right, contains all nodes and relationships needed to describe an application model. In the center, the composition of a sample application is shown, consisting of three VM types, the VM information about these VMs and the owner/user of the application. The GMF editor generates two files, one responsible for the graphical representation and the XMI description of the application model that can be uploaded to the social networking platform.

Chapter 4

Evaluation

This chapter describes the evaluation of the social networking platform focusing on scalability, the topic classification mechanism, and usability of its user interface.

4.1 Scalability

This section describes the evaluation of two different implementations of our system architecture. The first implementation adds more than one memcached instances in layer 2 (Figure 3.1). The second implementation features more than one social network engines in layer 1.

In order to measure the response time (RT) the Apache JMeter application [72] is used. The Apache JMeter is an open source benchmark designed to test functional behaviour and measure performance, targeting web applications. Notably, the RT measured by JMeter may not be the real one, because the JMeter measures the elapsed time from just before sending the request to just after the last response from the server has been received. As a result, the time to render the web page to the client web browser and the execution time of JavaScript code is not measured. Because those two time intervals are client limited and depend on client performance and on which web browser is used, they are excluded from the following performance test benches. For the next experiments, a specific web page will be used. This page does not use AJAX calls, so as not to interfere with the results. Therefore, the RT measures the time from just before JMeter sends the request to just after the last response is received. During this measured time interval, the Social Network engine performs the following actions:

I The social network engine sends a request to CDO client for the application execution model.

II The CDO client forwards this request to CDO server.

III The CDO server queries the MySQL repository of application models and executions, and finally gets the executions results.

IV CDO server forwards the results through the CDO client to the social network engine.

V Finally, the social network engine sends queries to the social network database in order to get all the necessary social features for this application page.

The presented system architecture was deployed on Amazon EC2 [82]. We measured and present the system CPU utilization and response time of the SNP.

4.1.1 Scaling the caching tier

By adding a memcached node at the system architecture, the social network engine first asks the memcached node if it has the tuples that the SN engine needs. So steps (*I* to *V*) are not carried out if the memcached node has cached the values requested by the social network engine. The loop through CDO client - CDO server and the repositories is bypassed.

State	# of Queries
Fresh start	1938
Fresh Query	15182
Cached Query from CDO	251
Cached Query from memcached	147

Table 4.1: Number of queries from social network and CDO server repository

For the following experiments all the memcached nodes are warmed up and have already cached all the needed CDO and Social entities information except for some initialization queries of Elgg framework. Furthermore the CDO server has been warmed up after a fresh restart. As the Table 4.1 shows, the starting process of the CDO server produces 1938 queries to MySQL database. The *fresh query* for an application model (both social information and executions) produces 15182 queries to MySQL database. The CDO server caches the results, so a second query for this application model produces 251 queries, most of which are the queries for the social information of the application. Introducing memcached, if the request for the application model is cached, the queries to database are lowering to 147. Those 147 queries are basically queries about the configuration of Elgg social networking engine and the sessions of the users.

The test performed with the following loads: (L1) ten users request *two* applications, (L2) ten users request *four* applications and (L3) ten users request *eight* applications. All three Loads run consecutively one hundred times each. Those Loads request applications, which have ten execution rows pulled from the repository of applications models and executions, and about one hundred queries to the social network database. In this experiment we kept the following components of the system constant: the Elgg front-end Apache2 server, the social network

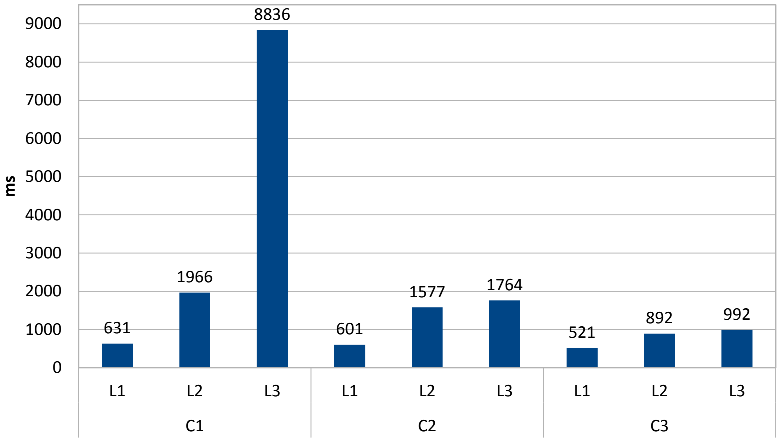


Figure 4.1: The average response time for all configurations

database, and the CDO server - client communication but we increased the number of memcached nodes. Figure 4.1 shows the average response time (RT) in milliseconds (ms) with the following system configuration(C): (C1) no memcached node, (C2) one memcached node and (C3) two memcached nodes have added to the system architecture.

Going from C1 to C3 and specifically for L3, the RT is reduced by 80,4% in C2 and by 88,78% in C3. As the Figure 4.1 shows, in the first configuration C1, the L3 takes 8836 ms, an RT which is definitely prohibitive for web applications. Introducing more memcached nodes at C2 and C3 the RT is decreased dramatically at 1764 ms at L2 and at 992 ms at L3. Going from C1 to C2, the 80,4% reduction of RT is due to the introduction of memcached node and bypassed the steps I - V, which are mentioned previously. Going from C2 to C3, the RT reduced by a factor of 43,77%. The reduction of RT is achieved by adding more memcached nodes, which results to more CPU cores introduced to the system, as described below.

Component	VM type
SN engine, CDO client	t1.micro
memcached	t1.micro
repositories, CDO Server	m1.xlarge
jmeter	m1.large

Table 4.2: VM resources

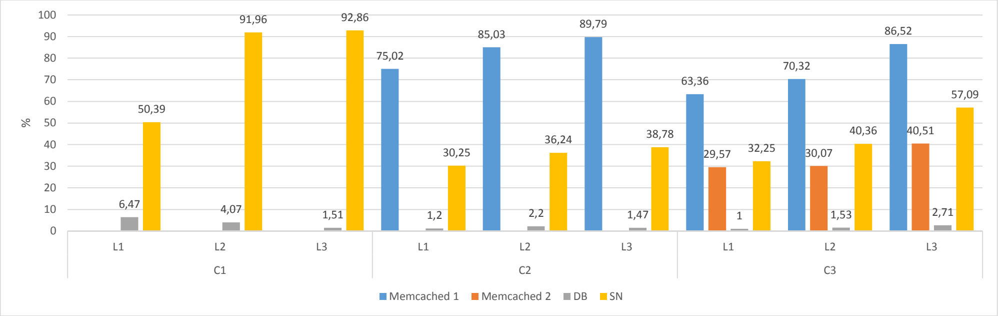


Figure 4.2: The average CPU utilization for all components

The CPU utilization is measured using the sysstat tool [73]. We measured the CPU utilization for all the VMs running the experiment. The information about the VM resources is listed in Table 4.2. The social network engine and the CDO Client were running at t1.micro instance. The MySQL (repositories) and the CDO Server were running at m1.xlarge. The average CPU utilization is shown in Figure 4.2. At the simple configuration C1, even in small loads such as L1, the SN engine reached 50,39% CPU utilization. In the medium load L2 and big load L3 the SN Engine is kneeled down to 91,96% and 92,86%. This big consumption of CPU was due to all the initialization process that Elgg social network engine has to do for each request and due to CDO server queries. Also, the CPU bottleneck of Elgg engine is responsible for the slow response time.

Moving from configuration C1 to C2, the CPU consumption went to memcached node. Thus, the social network engine was de-congested and the RT improved. However, for the big load L3 the memcached node reached 89,79%. To solve memcached CPU overhead, one more memcached node was added at configuration C3. This second memcached node shared the CPU overhead with the first memcached node and the RT improved furthermore.

For all three loads at C3, the first memcached node had more CPU utilization from the second by an approximately factor of 2,2. This difference between the two memcached nodes appeared due to the first node storing more popular key-value pairs than the other.

4.1.2 Scaling the Elgg engine tier

This section evaluates the horizontal scale of social network engine as described in Section 3.2.1. A memcached node was living between the social networking engines and the back end system. The VM resources were kept the same as in

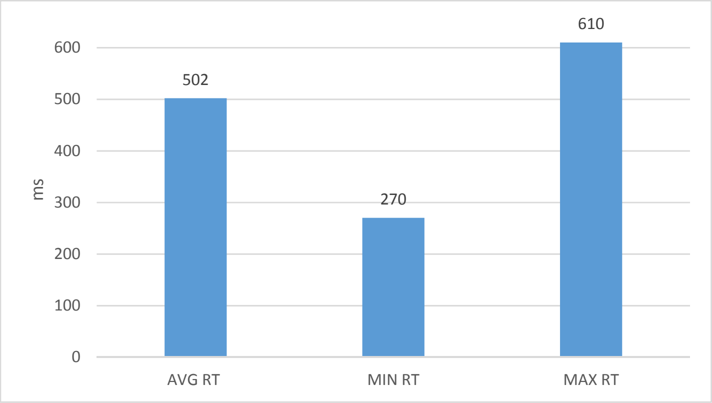


Figure 4.3: The response time for two social network engines

the previous experiment and are shown in Table 4.2. One more social networking engine instance was added with the same type as former SN engine.

So, the system architecture now consists of two social networking engines as front-end. At the back-end of the system we have: (1) one memcached node and (2) the CDO client - server, the social network database and the CDO repository. The two SN engines are deployed to a dedicated VM each. Furthermore, each SN engine has its own CDO client deployed with them. The memcached node is deployed on its own VM and the CDO Server, the social network database and the CDO repository are deployed on the same VM.

The Figure 4.4 shows the average CPU utilization for the memcached, the MySQL database (db) and the two SN Engines (SN1, SN2). The load for this experiment is the same as the previous load L3, which means we have ten users that request *eight* applications for one hundred times consecutively. The first SN engine has 43,2% CPU usage and the second one has 12,8% less. This difference is due to the first instance being deployed together with the NFS server and Apache Zookeeper on the same VM.

With only one SN engine the CPU utilization was 57,09% and now with two SN engines the CPU utilization is reduced to 43,2%. This reduction is due to the requests being distributed to two instances instead of the only one instance. The CPU utilization of the memcached node increased, but this can be solved by introducing more memcached nodes as the previous section describes. Furthermore, we can introduce more SN engines to support more heavy loads.

Since the load from one SN engine is now distributed to two SN engines instances, the response time improved for the Load 3, as shown in the Figure 4.3

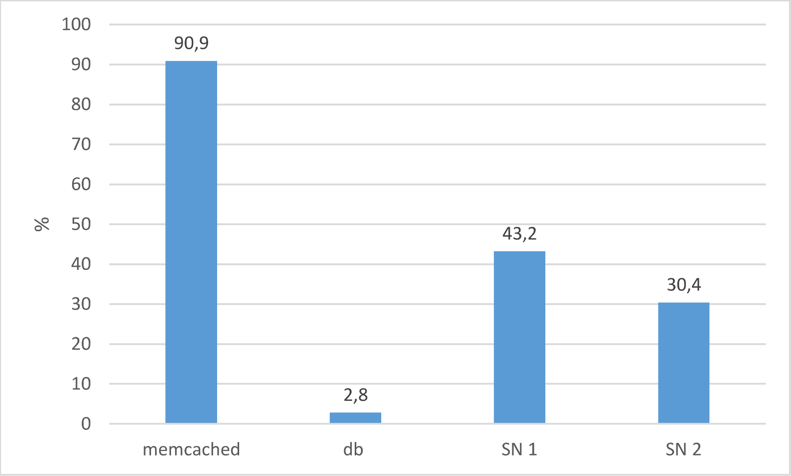


Figure 4.4: The CPU utilization for two social network engines

compared to the previous test-bench. The best achieved RT of previous architecture was 992 ms and with this architecture it reduced to 502 ms.

We can combine the two architectures together, meaning that we have more than one SN Engine and more than one memcached to support as many loads as we want.

4.2 Topic classification

This section evaluates the topic classification tool described in Section 3.3. The evaluated topic classifier had four different classes or, according to StackOverflow (SO) dialect, six tags, as are shown in Table 4.3. Those classes are: (1) *architecture and design*, (2) *scalability*, (3) *performance and optimization* and (4) *deployment*. This means that a new question can be classified in one of the previous four classes.

The *architecture* tag and the *design* tag comprised one class in our topic classifier because those two tags have similar meaning for the users of SO. Also, the *performance* and *optimization* tags defined another class for the same reason.

For the testing set, thirty questions from StackOverflow were used per class, in order to measure the topic classification accuracy. Those questions were different from the training set and were the most up-voted questions during the specific time period that this experiment was carried out. Each row in Table 4.3 shows a class as classified by StackOverflow users, and each column shows how our classifier classifies the question. For example, twenty nine questions about *architecture and design* were classified correctly, and only one was wrongly classified as *performance & optimization*. The misclassification however, was not an error of our classifier.

For the evaluation process, the title and the body of the questions are used to determine their class. The title of the question is very helpful for the determination

class / class	architecture & design	performance & optimization	deployment	scalability
architecture & design	**29**	0	0	1
performance & optimization	2	**23**	0	5
deployment	6	0	**22**	2
scalability	7	0	1	**22**

Table 4.3: Evaluation of topic classification

of the class because in same cases the SO users ask their questions in the title of the question and poses only code in the body of the question along with a small explanation about the code. So, without the title of the question the topic classifier can not determine correctly the class of the question. Furthermore, the code which was existing in the body of the questions was removed in order to not misguide our topic classifier.

Furthermore, the classes used in this evaluation have similar meaning to each other. SO users sometimes use those tags without knowing their meaning. Also, the scalability tag is too broad and can have keywords from the architecture and design class.

This shows that our tool can further be used by StackOverflow to mark new questions that are wrongly tagged or misguided. There is a trend among StackOverflow users to add as many tags as they can in order to attract the attention of other users, increase the views of their question and finally get their answers.

The true positive, which means a document is recognized in the correct class, according the Table 4.3 is 96. The false positive, which means a document is not correctly recognized, of this topic classification evaluation is 24. According to those precision metrics the accuracy or sensitivity of our topic classifier for this specific experiment is 80%.

For future work, this evaluation can be extended by imported questions and answers for other information repositories which have more similar terns to our social networking platform and with classes with different meaning to each other.

4.3 Requirements and user interface

The user interface (UI) of our social networking platform[1] is designed through an iterative process of several expert-based evaluations carried out in group sessions. To obtain additional feedback from non-experts, three additional user-based evaluation experiments were designed and carried out involving potential users and presented in detail in [6].

[1] The user interface (UI) design and usability evaluation was the result of a collaboration with the Human-Computer Interaction (HCI) laboratory of ICS-FORTH [6].

The author collaborated in the expert-based evaluations sessions focusing on the platform user interface design. He contributed in the UI usability evaluations through the implementation of the platform, and took active part by interviewing some of the participants in the third evaluation experiment.

The first experiment, carried out by Flexiant Ltd. aimed at assessing the overall look and feel of the network, the navigation mechanisms, as well as the design of fundamental functionality. The second evaluation experiment, carried out by a FORTH team consisting of HCI and CARV laboratory members, aimed to collect subjective results rather than performance metrics. It involved another set of 12 users who, after a brief introduction to the available facilities, were asked to use the interactive prototype [83] using the free exploration method of the Thinking Aloud protocol [84] and fill-in a questionnaire in order to rate and comment their experience.

The third evaluation experiment involved 15 participants guided through the interactive prototype of our social networking platform using specific scenarios. They were interviewed on their requirements and feedback following a semi-structured interview approach [84]. The evaluation session was carried out via Skype. Participants were recruited through European companies and organizations associated with the PaaSage EU project [15]. They were either developers or operations staff, thus within the target user groups of our social networking platform. Participants were not users of the platform; some however were familiar with the project's goals and objectives. Before the experiment, each participant was requested to fill-in a background information form and was sent an informed consent form, explaining all the recording and anonymity-ensuring procedures.

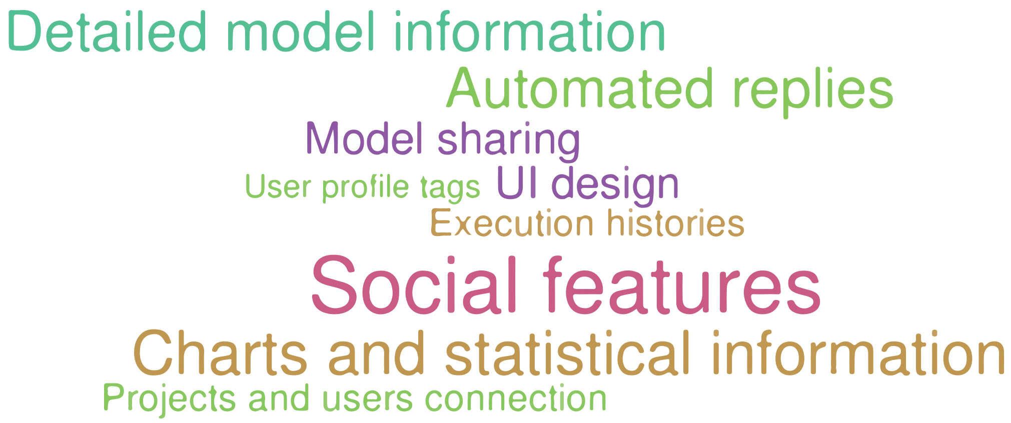


Figure 4.5: Features that were mostly liked by the users

Users in the third experiment were asked to identify up to three most-liked and three most-disliked features. Most-liked features presented in Figure 4.5 included: the employment of social features in a development environment; the use of charts and statistical information to represent data; the detailed model information that could be retrieved and the model execution histories; the automatically-generated hints provided by the network as replies in discussion topics; the concept of sharing one's models; the overall UI design; the direct connection between projects and users, and the user profile tags that allowed them to find users that would be interesting to connect with.

Those user interface evaluation experiments show that a platform that couples social networking features in community-building activities with DevOps requirements for application deployment, analyses of the execution histories, and automatically generated hints based on data analysis, is a helpful tool that people would like to use. Furthermore, those evaluation experiments show that the design and the implementation of our social networking platform was functional and easy to use.

Chapter 5

Conclusions and Future Work

This thesis describes and evaluates a scalable social networking architecture for DevOps engineers, with a special focus on cloud deployment specialists. The scalability of the architecture relies on the provisioning of multiple social networking engine instances at the front end, and/or several memcached nodes at the back end of the system. Users of the social networking platform can benefit from the community knowledge and from the CAMEL repository of application models and executions, to improve the configuration, the deployment and the optimization of distributed multi-cloud applications, tasks of major interest to cloud deployment specialists. Furthermore, we explored topic classification as a means to categorize community input and to better link it with existing content (past questions and answers, and the results of past queries over historical execution data).

The user evaluations and pilot use of our platform within the PaaSage project has helped improve our implementation. As future work, we believe that extending the integration of our social networking platform with more information repositories such as GitHub, can provide additional benefits to the DevOps community. A broader investigation of our topic classification system with a large user base is another promising avenue of future work.

Bibliography

[1] M. Loukides, *What is DevOps?* " O'Reilly Media, Inc.", 2012.

[2] J. Rossberg, "Collaboration," in *Beginning Application Lifecycle Management*. Springer, 2014, pp. 135–143.

[3] Chef, https://www.chef.io/, 2015, [Online; accessed 12-Aug-2015].

[4] "Ibm blue mix for developers," http://https://developer.ibm.com/bluemix/, 2015, [Online; accessed 24-May-2015].

[5] P. D. 2.1.2, "Model Based Cloud Platform Upperware," http://www.paasage.eu/images/documents/paasage_d2.1.2_final.pdf, 2014, [Online; accessed 18-May-2015].

[6] K. Magoutis, C. Papoulas, A. Papaioannou, F. Karniavoura, D.-G. Akestoridis, N. Parotsidis, M. Korozi, A. Leonidis, S. Ntoa, and C. Stephanidis, "Design and implementation of a social networking platform for cloud deployment specialists," *Journal of Internet Services and Applications*, vol. 6, no. 1, pp. 1–27, 2015.

[7] N. Ferry, A. Rossini, F. Chauvel, B. Morin, and A. Solberg, "Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems," in *Proceedings of CLOUD 2013: 6th IEEE International Conference on Cloud Computing*, L. O'Conner, Ed. IEEE Computer Society, 2013, pp. 887–894.

[8] F. Chauvel, N. Ferry, B. Morin, A. Rossini, and A. Solberg, "Models@ runtime to support the iterative and continuous design of autonomic reasoners." in *MoDELS@ Run. time*, 2013, pp. 26–38.

[9] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu, "Web services agreement specification (ws-agreement)," in *Open Grid Forum*, vol. 128, 2007, p. 216.

[10] C. Quinton, N. Haderer, R. Rouvoy, and L. Duchien, "Towards multi-cloud configurations using feature models and ontologies," in *Proceedings of the 2013 international workshop on Multi-cloud applications and federated clouds*. ACM, 2013, pp. 21–26.

[11] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro, *EMF: eclipse modeling framework.* Pearson Education, 2008.

[12] Eclipse, “CDO Model Repository,” http://projects.eclipse.org/projects/modeling.emf.cdo, 2015, [Online; accessed 29-July-2015].

[13] K. Kritikos, M. Korozi, B. Kryza, T. Kirkham, A. Leonidis, K. Magoutis, P. Massonet, S. Ntoa, A. Papaioannou, C. Papoulas, C. Sheridan, and C. Zeginis, “D4.1.1 – prototype metadata database and social network,” Accessed 8/2015, available from http://www.paasage.eu/images/documents/PaaSage-D4.1.1_final.pdf.

[14] A. Papaioannou and K. Magoutis, “An Architecture for Evaluating Distributed Application Deployments in Multi-Clouds,” in *Proceedings of 5th IEEE International Conference on Cloud Computing Technology and Science(CloudCom’13).* Bristol, UK: IEEE, 2013.

[15] PaaSage EU FP7 project, http://www.paasage.eu/, 2015, [Online; accessed 29-July-2015].

[16] D. Baur, S. Wesner, and J. Domaschka, “Towards a model-based executionware for deploying multi-cloud applications,” in *Advances in Service-Oriented and Cloud Computing.* Springer, 2014, pp. 124–138.

[17] D. W. Jorgenson and K. J. Stiroh, “Information technology and growth,” *American Economic Review*, pp. 109–115, 1999.

[18] “Social networks popular among programmers,” http://www.informationweek.com/wireless/social-networks-popular-among-programmers/d/d-id/1078472, [Online; accessed 12-Aug-2015].

[19] “Github,” http://github.com/, 2015, [Online; accessed 24-May-2015].

[20] “Gitter: The chat for github,” http://gitter.im/, 2015, [Online; accessed 24-May-2015].

[21] “Sourceforge,” http://sourceforge.net, 2015, [Online; accessed 24-May-2015].

[22] “Google code,” https://code.google.com, 2015, [Online; accessed 24-May-2015].

[23] “Codeplex: Project hosting for open source software,” https://www.codeplex.com, 2015, [Online; accessed 24-May-2015].

[24] “Stackoverflow,” http://stackoverflow.com/, 2015, [Online; accessed 24-May-2015].

[25] B. Vasilescu, V. Filkov, and A. Serebrenik, "Stackoverflow and github: Associations between software development and crowdsourced knowledge," in *Social Computing (SocialCom), 2013 International Conference on*, Sept 2013, pp. 188–195.

[26] "Ibm devops best practices," http://www.ibm.com/developerworks/devops/practices.html, 2015, [Online; accessed 24-May-2015].

[27] R. Sumbaly, J. Kreps, and S. Shah, "The big data ecosystem at linkedin," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. ACM, 2013, pp. 1125–1134.

[28] "Geeklist," https://geekli.st/, 2015, [Online; accessed 10-June-2015].

[29] "Snipplr," http://snipplr.com/, 2015, [Online; accessed 10-June-2015].

[30] "Masterbranch," https://masterbranch.com/, 2015, [Online; accessed 10-June-2015].

[31] "Dzone," http://www.dzone.com/, 2015, [Online; accessed 10-June-2015].

[32] "The code project," http://www.codeproject.com/, 2015, [Online; accessed 10-June-2015].

[33] J. Howe, "The rise of crowdsourcing," *Wired magazine*, vol. 14, no. 6, pp. 1–4, 2006.

[34] R. Sumbaly, J. Kreps, L. Gao, A. Feinberg, C. Soman, and S. Shah, "Serving large-scale batch computed data with project voldemort," in *Proceedings of the 10th USENIX conference on File and Storage Technologies*. USENIX Association, 2012, pp. 18–18.

[35] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," in *ACM SIGOPS Operating Systems Review*, vol. 41, no. 6. ACM, 2007, pp. 205–220.

[36] H. Barrigas, D. Barrigas, M. Barata, P. Furtado, and J. Bernardino, "Overview of facebook scalable architecture," in *Proceedings of the International Conference on Information Systems and Design of Communication*. ACM, 2014, pp. 173–176.

[37] "Bigpipe: Pipelining web pages for high performance," 2008, [Online; accessed 19-July-2008].

[38] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab *et al.*, "Scaling memcache at facebook." in *nsdi*, vol. 13, 2013, pp. 385–398.

[39] C. Lueninghoener, “Getting started with configuration management,” 2011.

[40] Bcfg2, http://www.bcfg2.org/, 2015, [Online; accessed 12-Aug-2015].

[41] CFEngine, http://www.cfengine.com/, 2015, [Online; accessed 12-Aug-2015].

[42] Puppet, http://www.puppetlabs.com/, 2015, [Online; accessed 12-Aug-2015].

[43] A. Tsalolikhin, “Summary, configuration management summit,” 2010.

[44] T. Delaet, W. Joosen, and B. Vanbrabant, “A survey of system configuration tools,” in *Proceedings of the 24th International Conference on Large Installation System Administration(LISA’10).* San Jose, CA: ACM, 11/2010, pp. 1–8.

[45] M. Fowler and M. Foemmel, “Continuous integration,” *Thought-Works) http://www.thoughtworks.com/continuous-integration*, 2006.

[46] “Travis continuous integration,” https://travis-ci.org/, 2015, [Online; accessed 12-Aug-2015].

[47] “Jenkins continuous integration.”

[48] C. D. Manning and H. Schütze, *Foundations of statistical natural language processing.* MIT press, 1999.

[49] A. Pak and P. Paroubek, “Twitter as a corpus for sentiment analysis and opinion mining.” in *LREC*, vol. 10, 2010, pp. 1320–1326.

[50] S. Verma, S. Vieweg, W. J. Corvey, L. Palen, J. H. Martin, M. Palmer, A. Schram, and K. M. Anderson, “Natural language processing to the rescue? extracting" situational awareness" tweets during mass emergency.” in *ICWSM.* Citeseer, 2011.

[51] A. Go, R. Bhayani, and L. Huang, “Twitter sentiment classification using distant supervision,” *CS224N Project Report, Stanford*, vol. 1, p. 12, 2009.

[52] E. Wong, J. Yang, and L. Tan, “Autocomment: Mining question and answer sites for automatic comment generation,” in *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on.* IEEE, 2013, pp. 562–567.

[53] A. Zookeeper, “Apache zookeeper,” https://zookeeper.apache.org/, 2015, [Online; accessed 20-May-2015].

[54] E. S. N. Engine, “Elgg social networking engine,” http://elgg.org/, 2015, [Online; accessed 19-May-2015].

[55] jQuery, “jquery,” https://jquery.com/, 2015, [Online; accessed 19-May-2015].

[56] M. Kolšek, "Session fixation vulnerability in web-based applications," *Acros Security*, p. 7, 2002.

[57] W. Burgers, R. Verdult, and M. van Eekelen, "Poster: Prevent session hijacking."

[58] J. L. Thames, "Comparing cross-site scripting vulnerabilities."

[59] "Bootstrap front-end framework," http://getbootstrap.com/2.3.2/, [Online; accessed 24-May-2015].

[60] E. McCormick and K. De Volder, "Jquery: finding your way through tangled code," in *Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications.* ACM, 2004, pp. 9–10.

[61] "Chart.js, simple, clean and engaging charts for designers and developers," http://www.chartjs.org/, 2015, [Online; accessed 12-Aug-2015].

[62] K. V. Natda, "Responsive web design," *Eduvantage*, vol. 1, no. 1, 2013.

[63] "Twitter Bootstrap," http://getbootstrap.com/, 2015, [Online; accessed 12-Aug-2015].

[64] D. Cochran, *Twitter Bootstrap Web Development How-To.* Packt Publishing Ltd, 2012.

[65] "Zeromq, the guide," http://zguide.zeromq.org/page:all, 2015, [Online; accessed 12-Aug-2015].

[66] "Rabbitmq, feature highlights," https://www.rabbitmq.com, 2015, [Online; accessed 12-Aug-2015].

[67] P. Sahni and A. Batra, "Network file system," *International Journal of Research*, vol. 2, no. 4, pp. 894–896, 2015.

[68] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "Zookeeper: Wait-free coordination for internet-scale systems." in *USENIX Annual Technical Conference*, vol. 8, 2010, p. 9.

[69] "The Zookeeper recipes," https://github.com/Gutza/php-zookeeper-recipes, 2015, [Online; accessed 12-Aug-2015].

[70] Memcache, "Memcache," http://memcached.org/, 2015, [Online; accessed 18-May-2015].

[71] "PHP Memcached library documentation," http://php.net/manual/en/book.memcached.php, 2015, [Online; accessed 12-Aug-2015].

[72] A. jMeter, "Apache jmeter," http://jmeter.apache.org/, 2015, [Online; accessed 19-May-2015].

[73] sysstat, "Performance monitoring tools for linux," https://github.com/sysstat/sysstat, 2015, [Online; accessed 19-May-2015].

[74] "Natural language processing with node.js," https://github.com/NaturalNode/natural/, 2015, [Online; accessed 1-July-2015].

[75] "Stack exchange application programming interface," https://api.stackexchange.com/, 2015, [Online; accessed 14-July-2015].

[76] M. F. Porter, "An algorithm for suffix stripping," *Program*, vol. 14, no. 3, pp. 130–137, 1980.

[77] S. Deterding, M. Sicart, L. Nacke, K. O'Hara, and D. Dixon, "Gamification. using game-design elements in non-gaming contexts," in *CHI'11 Extended Abstracts on Human Factors in Computing Systems.* ACM, 2011, pp. 2425–2428.

[78] J. Antin and E. F. Churchill, "Badges in social media: A social psychological perspective," in *CHI 2011 Gamification Workshop Proceedings (Vancouver, BC, Canada, 2011)*, 2011.

[79] "Graphical Modeling Framework," https://wiki.eclipse.org/Graphical_Modeling_Framework, 2015, [Online; accessed 30-July-2015].

[80] "Cookbooks Site API," https://docs.chef.io/api_cookbooks_site.html, 2015, [Online; accessed 30-July-2015].

[81] E. Papoutsakis, "Reducting the complexity of model-driven design and deployment of multi-cloud applications." Master's thesis, Computer Science Dep. of Un. of Crete, 2015.

[82] "Amazon elastic compute cloud," https://aws.amazon.com/ec2/, 2015, [Online; accessed 12-Aug-2015].

[83] R. A. Virzi, J. L. Sokolov, and D. Karis, "Usability problem identification using both low- and high-fidelity prototypes," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '96. New York, NY, USA: ACM, 4/1996, pp. 236–243. [Online]. Available: http://doi.acm.org/10.1145/238386.238516

[84] P. Jordan, "An introduction to usability," 1998.