©2010 International Journal of Computer Applications (0975 - 8887)
Volume 1 – No. 21

Domain Specific Languages

Aruna Raja
Usha Mittal Institute of Technology,
SNDT Women's University,
Santacruz (W), Mumbai, 400049

Devika Lakshmanan
Usha Mittal Institute of Technology,
SNDT Women's University,
Santacruz (W), Mumbai, 400049

ABSTRACT
To match the needs of the fast paced generation, the speed of computing has also increased enormously. But, there is a limit to which the processor speed can be amplified. Hence in order to increase productivity, there is a need to change focus from processing time to programming time.

Reduction in programming time can be achieved by identifying the domain to which the task belongs and using an appropriate Domain Specific Language (DSL). DSLs are constrained to use terms and concepts pertaining to an explicit domain making it much easier for the programmers to understand and learn, and cuts down the development time drastically.

In this paper, we will understand what a DSL is; explore a number of DSLs spanning various phases of software development life cycle in terms of features that elucidates their advantages over general purpose languages and perform in depth study by practically applying a few open source DSLs: 'Cascading', Naked Objects Framework and RSpec.

Categories and Subject Descriptors
D.1.m [**Programming Techniques**]: Miscellaneous

General Terms

Performance, Design, Languages

Keywords

Domain Specific Language. Fluent Interfaces, Method Chaining, RSpec, Cucumber, RGen, Graphviz, 'Cascading', Naked Objects Framework, Maestro, ScalaModules, Make, Rake, Twill, Twist, SmartFrog, Chef, EC2 Deploy Framework

1. INTRODUCTION
Domain Specific Language (DSL) is a computer language that is targeted to a particular kind of problem, rather than a general purpose language that's aimed at any kind of software problem [1].

The concept of DSLs has been around for a long time in the form of mini-languages such as shell utilities like awk, sed etc in the Unix community. These languages specialized in a particular domain for reduced lines of code. A number of tools like spreadsheet, HTML, SQL, CSS that have been in widespread use are also DSLs. But the term has only gained popularity recently with the advent of Domain-specific modeling.

A DSL is simple and concise with the expressive power focused on a particular problem domain. It is custom built to be very intuitive and fluent for a domain expert (even non- programmers) to use, validate, modify and even write DSL programs. It allows one to construct efficiently and quickly complete applications for that domain, thus reducing programming time and increasing productivity. These advantages of DSLs over general purpose languages (GPLs) are making DSLs very popular and the design and implementation of DSLs an intensive area of research.

The goal of DSLs is to provide a highly effective interface that allows users to interact with their application. They try to screen away the internal details of the application and let the users work with familiar terms and concepts of their domain. Classic example of a DSL that has been used widely for a long time is that of SQL for writing database queries. For a given problem, the users need to only think about the SQL queries and commands required and not the actual underlying operations that work on the database.

Features of DSL

- It is designed to be simple in order to reduce the learning time.
- It is built on the domain user's vocabulary.
- The syntax provided hides the inherent programming aspect of the application from the client.
- In spite of an increased startup cost, DSL - based methodology renders a lesser Total Software Cost compared to a conventional methodology.

2. NEED FOR DSLS
The following reasons have lead to the need to create and use DSLs:

- Creating a domain-specific language can be worthwhile if the language allows particular type of problems or solutions to them to be expressed more clearly than pre-existing languages would allow, and the type of problem in question reappears sufficiently often.
- In order to reduce development time, tools with reusable code libraries are required. Repetitive tasks to be performed are readily defined in DSLs with custom

libraries whose scope are restricted to the domain and hence need not be written from scratch each time.

- There is need for a solution that empowers experts with the power to specify the logic of their applications and maintain it at the same time as and when requirements change. Domain specific languages provide such solutions that help domain experts to easily comprehend and create code for their application. The self documenting feature of DSLs supports it further.
- It is difficult to map conceptual model of solution into mainstream programming language as most time is spent in finding ways to express natural language concepts in terms of programming level abstractions (e.g. classes, methods, loops, conditionals, etc.). The mapping to DSLs becomes much easier and straightforward because DSLs make use of terms and concepts dealt in the specific domain instead of being forced to translate ideas into notion that a GPL is able to understand.

3. DOMAIN SPECIFIC LANGUAGES vs GENERAL PURPOSE LANGUAGES

Domain specific languages have a number of advantages over general purpose languages such as Java and c++, some of which are listed below:

- The scope of a DSL is only up to a specific domain. It therefore allows any domain expert to use it, in contrast with general purpose language that requires core programming capabilities in order to develop applications.
- Domain specific languages are very expressive i.e. their syntax is readable and easily understandable.

  front_door.paint(3,:red).dry(30).close

  For example, the above piece of code written in Ruby gives a clear idea to anyone about what it is trying to implement.
- DSLs reduce complexity by screening away the internal complex operations of the system. GPLs would require manual coding of every detail that becomes cumbersome and time consuming.
- DSLs are more productive as they need lesser programming time compared to GPLs.
- Domain specific languages support standardization wherein the underlying implementation can be changed without the need to change the code. For example, HTML is browser independent and can work on all kinds of browsers.

4. FLUENT INTERFACES

Fluent Interface is a term coined by Eric Evans and Martin Fowler that is used to describe objects that expose an interface that flows, and is designed to be readable and concise. They allow one to express the code in the terms of the domain being worked on. It is basically a design pattern followed to create API for DSLs.

It adds on to readability using the following features:

Method chaining

Method chaining allows each property to be set through a method call and then have that method return a reference to it so as to continue on with the next method call. A simple example for method chaining would be

string user = new StringBuilder().Append("Name: ").Append(emp.Name).AppendLine().Append("EmpCode: ").Append(emp.Code).AppendLine().ToString();

Factory classes

Used in cases where there is a need to build up a series of related objects. These classes provide methods to manufacture the instances required.

For example, in NUnit 2.4 the following can be written:

Assert.That(result, Is.EqualTo(4)); [2]

Where 'Is' class is the factory class and EqualTo is one of the factory methods it contains. These methods specify the constraints for evaluation in NUnit.

Named parameters

Languages like Smalltalk and C# 4.0 contain a feature called Named parameters which provide a way to include additional "syntax" in a method call which in turn improves readability [2].

For example, consider a user defined Set () method which takes a user name and password for setting up a new login Id.

myLogin.Set("DSL","Explore");

With Named Parameters the above can be represented more precisely as:

myLogin.Set(User:"DSL", Password:"Explore");

5. TYPES OF DSLS

There are two types of DSLs –Internal DSLs and External DSLs

In case of an internal DSL, the aim is to reuse and extend a given host language. The key feature is that all the infrastructure of the host language is inherited by the DSL. The DSL is constructed / embedded as a library of functions written in the host language. Constructed in this way, the DSL can be easily extended by simply adding a new function to the library. Internal DSLs are generally designed using dynamic languages such as Ruby that contain meta-programming features. Meta programming is the creation of computer programs that writes or manipulates other programs (or themselves) as their data, or that do part of the work at compile time that is otherwise done at run time. In many cases, this allows programmers to get more work done in the same amount of time as they would take to write all the code manually. Common examples of internal DSLs are Ruby on Rails and RSpec using Ruby, 'Cascading' and Smartfrog with Java as their base language.

For external DSLs such as SQL and Make, the idea is to build a DSL from scratch. Hence, grammar needs to be created for the language that requires additional efforts in the form of a compiler that is required to parse and process the syntax and map it to the semantics.

An advantage of internal DSL is that the compiler or interpreter of the base language is reused as it is. Because the programmer takes the effort to define the grammar for an external DSL, that effort also serves to validate the syntax. This is harder to do with an internal DSL because the code is often processed dynamically. Extensive error checking and validation has to be done. Furthermore, it is constrained by the host language's syntax and structure. In contrast, an external DSL gives a lot of flexibility, but it involves the overhead of increased compile time.

6. DSLS IN SOFTWARE DEVELOPMENT LIFECYCLE

The various phases in SDLC can be viewed as separate technical domains and many in the DSL forefront are actively involved in creating one for handling the basic functions performed in these domains.

6.1 Specification phase

It is of utmost importance as any miscommunication of requirements happening in this phase could lead to an outcome not as per the client's needs. To deal with this situation an "outside-in" methodology called Behaviour Driven Development (BDD) can be employed. It starts at the outside by identifying desired outcomes and then works out on the set of features that will help to achieve them. To simplify these tasks in the specification domain DSLs like RSpec and Cucumber have been created and are described below.

6.1.1 RSpec

Ruby based DSL RSpec achieves BDD by creation of 2 separate files, one containing the specification test cases and other comprising enough code to pass through the same. It gives the output in terms of the number of executable examples and failures. These executable examples are expected behaviour of the system listed using the method 'it' and any deviation to achieve them is reflected as a failure. They are mainly used to write unit tests. It is an open source tool; its gem can be readily installed and included into the ruby library for utilization.

6.1.2 Cucumber

Cucumber is a tool that can execute plain-text documents as automated functional tests. The stories are specified using Given-And-When-Then steps making it declarative enough for even a non-programmer business specialist to understand. It can be of interest as can be used for integration testing. This tool written in ruby is also quite capable of testing code in other programming languages with the aid of some extra tools.

Cucumber stories do not substitute for RSpec unit tests. The two go hand in hand. The specs for unit tests tend to drive development of components where as the stories provide a good test of the application as a whole from the user's perspective.

6.2 Design phase

The primary objective of the design phase is to create a design that satisfies the agreed application requirements. DSLs that support auto generation of designs through small pieces of code have been created.

6.2.1 RGen

The Ruby based RGen is a modeling and generator framework. It provides support for dealing with models and metamodels, for defining model transformations and for code generation. Following the Ruby design principles, it is lightweight and flexible and supports efficient development by providing means to write concise, maintainable code, thus allowing for efficient development and simple deployment [3]. RGen uses its object-oriented programming for representation of models and metamodels: objects to represent model elements and classes for metamodel elements. In model transformation, an instance of one metamodel is converted into an instance of another metamodel. RGen also supports code generation wherein model is transformed to textual output. RGen, being dynamically typed has certain disadvantages such as missing compiler checks and editor support. They are compensated by means of a more intensive unit testing and using those dynamic language features that makes use of editor support difficult to build. Typical applications include code generators, prototyping tool in automotive industries, tools for building and manipulating models, mostly in XML.

6.2.2 Graphviz

Graphviz is software for automatically drawing graphs and networks using a very compact code as compared to other graph visualization tools like SVG. Hence there is enormous reduction in programmer's time.

It supports in-built options for colour, fonts, line styles, custom shapes, etc making it much user friendly. One of the many applications is Instaviz which is a graph sketching application for the iPhone using Graphviz libraries for rendering.

6.3 Implementation phase

DSLs have been created for implementing tasks from various domains like distributed computing, object-oriented programming, building software etc.

6.3.1 Cascading

'Cascading' is a java based DSL created by Chris Wensel for faster implementation of Map/Reduce problems on Hadoop cluster. It helps to overcome problems like thinking in terms of Map/Reduce as well as executing larger number of Map/Reduce jobs in sequence by simply using 5 core components namely: Tuple, Tap, Pipe, Flow and Cascade.

Tuple is used to represent the records from input files. Tap is a resource like data files on local, distributed, or Hadoop file systems. Pipe performs operations on each record and is of three types: Each, Group and Every. Flow connects the pipe assembly to source tap and sink tap on either sides. Cascade is an optional stage which merges multiple flows.

6.3.2 Naked Objects Framework

Naked Objects Framework (NOF) is a DSL which was originally created for fast prototyping of object- oriented applications. Since the introduction of persistence, complete applications like the Irish government's social benefits program has been created. It is available as a commercial application development platform in .Net and an open source development platform in Java. It requires mere creation of behaviourally rich domain objects and the other layers like User Interface, persistence layer etc are auto-generated by NOF. Behaviourally rich domain objects simply need defining of the parameters and actions associated with the instance of objects and the various services required. The entire NOF can be summarized by the conceptual diagram shown below:

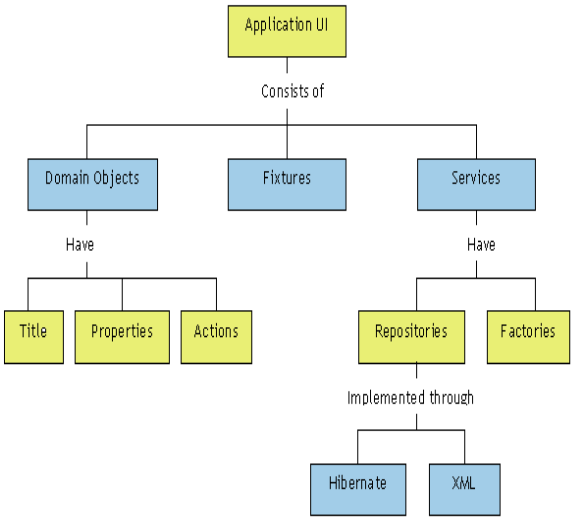

Fig 1: Conceptual Diagram of NOF

The various components of NOF as shown in figure 1 are as follows:

Title- Name associated with each instance of Domain object.

Properties- Parameters of the Domain objects.

Actions- Operations needed to be performed on each instance of Object.

Fixtures- Contains data to be launched each time the application is run during demos.

Services- It defines functions that cannot be applied to single instance. Eg: Find developer, Show all etc

Repositories- Complex services required to be defined by user. These are implemented through object stores like Hibernate and XML.

Factories- Services like object creation is built in NOF.

NOF supports two versions for interface: Drag and Drop and Web view. Drag and Drop is the desktop version used for standalone operations. Web view can be used over the web browser.

6.3.3 Maestro

Maestro is an internal DSL with actor based concurrency on .NET platform used for writing distributed applications. Maestro is a subset of language features which allow for easier isolation and concurrency within that isolation.

Built upon Concurrency and Coordination Runtime (CCR), it is an Actor oriented language with constructs like Domain, Agents, Channels, Schemas and Networks. This adds on features like process isolation, message passing, fault tolerance and loose coupling required for handling concurrency.

6.3.4 ScalaModules

ScalaModules is a DSL for Scala-based OSGi development. The aim of ScalaModules is to employ the power of the Scala programming language to ease OSGi development. Scala is statically typed dynamic language and hence ScalaModules' OSGi code will be more intuitive and concise as well as less verbose and involved compared to Java-based development [4].

The Open Services Gateway Initiative (OSGi) is an independent, non-profit corporation working to define and promote open specifications for the delivery of managed services to networked environments, such as homes and automobiles. These specifications define the OSGi Service Platform, which consists of two parts: the OSGi framework and a set of standard service definitions which are defined by ScalaModules [5].

Scala and OSGi both aim at very important concerns about software development: ease and reduced complexity. Scala operates at the bottom level of a programming language and OSGi at the higher level of a module system. Hence it is natural to combine those two to get the best out of both. It fully supports interoperability as Scala compiles to the usual Java byte code.

6.3.5 Make

'Make' is a dependency tracking build utility tool for executable programs and libraries. This external DSL is a declarative programming language where the build process is dependent on a given platform. Because of its compatibility with UNIX platforms it has attained wide spread popularity. 'Make' builds object files from the source files and links the object files to create the executable. If a source file is changed, only its object file needs to be compiled and then linked into executable instead of recompiling all source files. Since 'Make' utility is not an executable program on its own, a Makefile is used to derive target program from each of its dependencies. But, Make is error-prone as it requires programmers to manually track all dependencies between files in a project.

6.3.6 Rake

Rake is a Ruby variant of 'Make' and used to build simple build scripts. But unlike 'Make' it is an internal DSL which uses ruby syntax. Rake is a software build tool created by Jim Weirich in order to define tasks, do dependency task tracking with an additional feature which rebuilds only modified source files since last compilation. The third feature however is not required as Rake is an interpreted language. Rake makes use of a Rakefile as a reference to all the tasks and dependencies. It can be used for automating tests, generating HTML documents, cleaning up

generated files not required for the project, to build a gem package etc.

6.4 Testing phase

DSLs have been created to perform different types of tests on various applications.

6.4.1 Twill

This DSL written using python as host language aims at testing web applications. The simplest way to test web sites is to write one or more Twill scripts and then simply run from the command-line. It can be used for both unit as well as stress testing. The assertion commands built into Twill (code, find and notfind) should be enough for testing web sites that use straight HTML and forms. For more complicated Javascript-intensive web sites, a tool such as Selenium might be more appropriate [6].

6.4.2 Twist

Twist is an IDE created by ThoughtWorks Studios for functional testing of web and java-applications. It can also run regression, smoke, performance type of tests. The tool provides a single platform for documenting user stories, capturing executable requirements, developing, maintaining, running and reporting on functional tests. This tool implemented on Eclipse platform provides support for DSLs. Lines in DSL map into the underlying test automation using Selenium and Frankenstein. Tags can be associated with the tests to perform filtering for running subsets of test.

6.5 Deployment phase

After the system has been tested during the Testing Phase, and accepted by the user, the system is installed and made operational in a production environment, in accordance with the requirements. This has been made easier for large distributed systems through DSLs.

6.5.1 SmartFrog

SmartFrog is a powerful and flexible Java-based software framework used in the domain of configuration-driven systems for configuring, deploying and managing distributed software systems [7]. It uses notations to specify which applications are running where, how each component configured and their lifecycle are sequenced and how they are related. This HP Labs system uses a declarative template language for describing deployments. It has multiple software components running across a network of computing resources, where the components must work together to deliver the functionality of the system as a whole. There is no central server; you can deploy a .SF configuration file to any node and have it distributed to peer nodes according to the distribution information contained inside the deployment descriptor itself. It is critical that the right components are running in the right places, that the components are individually and collectively configured, and that they are correctly combined to create the complete system. This profile fits many of the services and applications that run on today's computing infrastructures. It is an open source tool and can be used by anyone for the purpose of deployment. Recent version of the tool has been successfully run on both Linux and Windows. Currently used by HP Labs for infrastructure and service automation.

6.5.2 Chef

Chef is a Ruby based DSL interpreted by chef clients working under chef servers. It uses a pure Ruby DSL for writing configuration "recipes". Clients authenticate themselves using an OpenID. Synchronization of needed resources and libraries are done automatically. These resources are used for configuring the client node, a process called convergence [8]. Chef can be used as a client-server tool, or used in "solo" mode. Recent version of the tool has been successfully run on Linux.

Both support mutual authentication. Encryption is fully supported by SmartFrog but partially by Chef.

6.5.3 EC2 Deploy Framework

EC2 Deploy is an open source framework for deploying an enterprise Java application on a set of Amazon EC2 servers. This deployment tool is written in Groovy.

EC2 (Elastic Cloud Compute) provides on – demand computing. It is one of the pay per use services managed by Amazon. It has a virtualized computing environment with server instances managed by a web service API.

EC2 Deploy describes clusters in terms of servers and location of web applications. It provides a methodology that easily launches EC2 instances, configures servers, deploys web applications and performs JMeter tests on clusters.

7. EXPERIMENTS WITH DSLS

In order to practically apply a few DSLs, small applications were created using ‘Cascading’, Naked Objects Framework and RSpec respectively.

7.1 Brand name change application using ‘Cascading’

The product created can be used by companies at time of brand name change. In such a scenario the new name needs to replace the old one in all the web pages associated with the company so as to adopt the new brand name. For huge firms the number of places where such a change will be encountered would be huge and the task might consume substantial amount of time if done manually. In an attempt to reduce this time, the concept of distributed computing and Map/Reduce can be utilized.

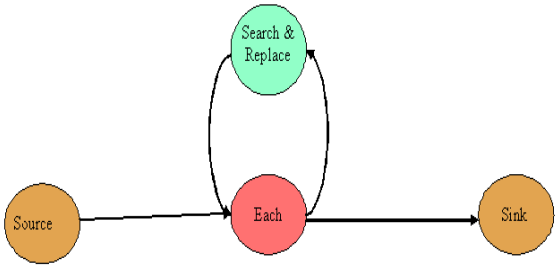

Figure 2. Pipe diagram for the cascading application

The product is developed using the 'Cascading' DSL created over a distributed Hadoop cluster. The application makes use of a single 'Each' pipe applying the 'search and replace' operation on each record from the input web pages as shown in figure 2. Thus, the user is not required to think in terms of Map/Reduce resulting in drastic reduction of development time.

7.2 Defect Tracking System using Naked Objects Framework

This business system has been designed using Naked Objects Framework to manage the various defects that have been logged by the QA team of the organization. Two domain objects namely Developer and Defect have been created along with the associated parameters, actions and services. They are displayed on the UI upon which the user can carry out various operations. Persistence has been achieved via XML object store. Through fixtures inbuilt data gets loaded onto the UI every time the system runs in the transient mode.

The Developer object in figure 3 allows the user to include any new developer into the system and conduct various operations like finding developers by name or id and also to delete them from the system if required.

The Defect object as shown in figure 4 helps to register any new Defects identified. The next step would be to allocate a developer to the defect from the set of developers in the system. This is done either by using the Drag and Drop mechanism in DND viewer or by using the Drop down menu in the HTML viewer.

NOF supports validation through a few built in methods and registered annotations that have been used to validate the parameters of Defect and Developer objects.

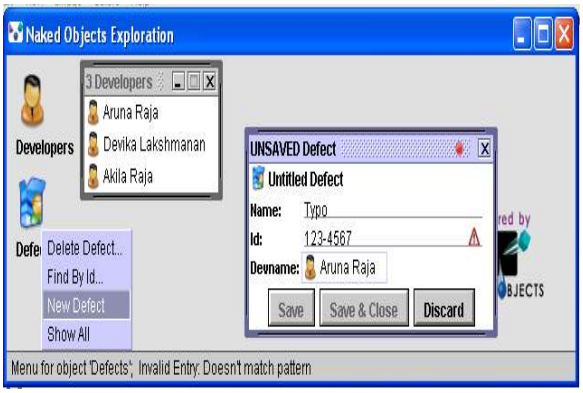

Figure 3. Developer Domain Object

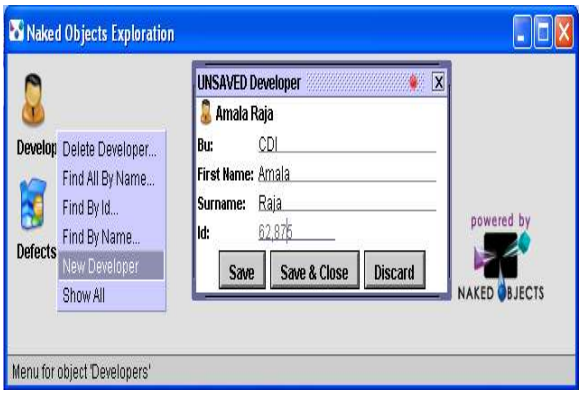

Figure 4. Defect Domain Object

7.3 Using RSpec to create specifications for a Finite State Machine (FSM)

An application defining the specifications for a FSM has been written using the following steps [9]:

- Write a test. This test describes the behavior of a small element of the system.
- Run the test. The test fails because the code for that part of the system has not been built yet. This step tests the test case, verifying that the test case fails when it should.
- Write enough code to make the test pass.
- Run the tests and verify that they pass.

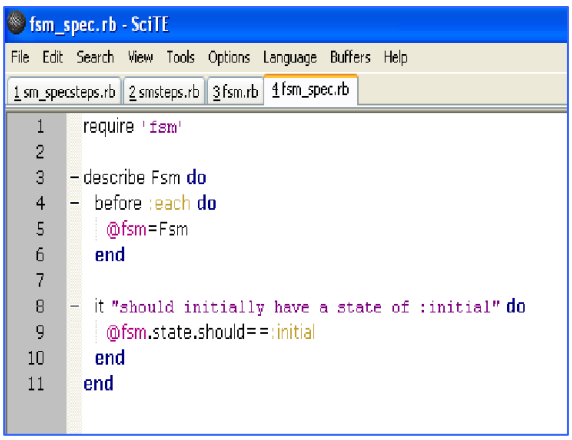

Figure 5.Specifications file for FSM in RSpec

The specifications file in figure 5 contains the various specs such as the valid states and events of the machine and correct transitions upon different triggering events. These specifications describe the basic requirements that need to be satisfied by the code to be of acceptable level. Since the code is written to pass these specifications, they tend to capture the client's requirements completely and have minimal errors. As testing is performed along with the development, testing is not compromised as deadlines approach closer.

8. CONCLUSION

In today's world, every possible domain requires the use of computers either to perform calculations, store information or write programs and hence a number of Domain Specific Languages have been designed to serve the need. Since the focus of each DSL is just a domain, it can be used by any specialist of that domain to write applications without the need to have any core programming skills.

The concept of DSLs has been around in the Unix community since long in the form of mini-languages with the focus being reduction in lines of code. However, DSLs now have additional features to aid readability, reusability and maintainability due to the introduction of object oriented and dynamic languages.

DSLs have really changed the way of building software and many in the DSL forefront see the future of programming as writing DSLs for specific platforms, frameworks (i.e. Rails), or the actual language used for writing the code in a specific business domain (i.e. Domain Driven Development). With the promising productivity boost and smaller code base feature, the future for DSLs seems to be brilliant and a study on this subject is worthwhile.

9. ACKNOWLEDGMENTS

We would like to express our gratitude to Usha Mittal Institute of Technology and Patni Computer Systems Ltd. for providing us with this brilliant opportunity to take up this project activity as a part of the curriculum.

10. REFERENCES

[1] M.Fowler, "Domain-specific language ", Oct. 2008 [Online]: Available: http://en.wikipedia.org/wiki/Domain_specific_languages. [Accessed: Jan. 16, 2009]

[2] Bevan, "Fluent Interfaces – Method chaining", Nov. 2008 [Online]. Available: http://stackoverflow.com/questions/293353/fluent-interfaces-method-chaining [Accessed: January 23, 3009]

[3] M.Thiede, "RGen: Ruby Modelling and Code Generation Framework", Feb. 2009 [Online]. Available: http://www.infoq.com/articles/thiede-ruby-modelling. [Accessed: Feb. 26, 2009]

[4] "Scala DSL to ease OSGi development", March 2009 [Online]. Available: http://wiki.github.com/hseeberger/scalamodules [Accessed: March 11, 2009].

[5] H.Cervantes and R.S.Hall, "OSGi in a nutshell", March 2004 [Online]. Available: http://gravity.sourceforge.net/servicebinder/osginutshell.html [Accessed: March 11, 2009].

[6] G.Gheorghiu, "Agile Testing Web app testing with Python part 3: twill", Sep. 2005 [Online]. Available: http://agiletesting.blogspot.com/2005/09/web-app-testing-with-python-part-3.html [Accessed: June 05, 2009]

[7] "Homepage – SmartFrog", Feb. 2009 [Online]. Available: http://wiki.smartfrog.org/wiki/display/sf/SmartFrog+Home. [Accessed: Feb. 11, 2009]

[8] C.Martin, "Chef Configuration and Provisioning Tool Announced", Jan. 2009 [Online]. Available: http://www.infoq.com/news/2009/01/chef-management-tool-announced. [Accessed: Feb. 5, 2009]

[9] B.Tate, "Behavior-driven testing with RSpec", Aug. 2007 [Online]. Available: http://www.ibm.com/developerworks/web/library/wa-rspec/. [Accessed: Feb. 19, 2009]